ISTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY

POSITIONAL KINEMATICS ANALYSIS OF THE 6-3 STEWART PLATFORM MECHANISM USING HEURISTIC ALGORITHMS

M.Sc. Thesis by

Elmas ANLI, B.Sc.

Department: Interdisciplinary Programs

Programme: Aeronautical and Astronautical Engineering

MAY 2005

İSTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY

POSITIONAL KINEMATICS ANALYSIS OF THE 6-3 STEWART PLATFORM MECHANISM USING HEURISTIC ALGORITHMS

M.Sc. Thesis by
Elmas ANLI, B.Sc.
(511031005)

Date of Submission: 5 May 2005

Date of Defence Examination: 30 May 2005

Supervisor (Chairman): Assoc. Prof. Dr. İbrahim ÖZKOL

Members of the Examining Committee: Senior Lecturer Dr. T. Berat KARYOT

Assoc. Prof. Dr. Haydar LİVATYALI

MAY 2005

ACKNOWLEDGEMENTS

Stewart platform mechanisms are used in various areas of engineering, including flight simulators. It is important to be able to solve their forward kinematics accurately since it is important to know the position and orientation of the mechanism once the lengths of its legs are known.

I wish to express my gratitude to my supervisor Assoc. Prof. Dr. İbrahim Özkol for supporting and encouraging me at every stage of my thesis. I also wish to express my gratitude to Dr. Sait N. Yurt for his patience in helping me and answering my questions at every stage of my thesis.

Last, but not the least, I thank my family and friends for being there whenever I needed them.

May 2005 Elmas Anlı

CONTENTS

ABBREVIATIONS

SPM	: Stewart Platform Mechanism
NN	: Neural Network
IK	: Inverse Kinematics
FK	: Forward Kinematics

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

M	: Mobility of the mechanism
n	: number of links in the system
f_1	: number of joints with one degree of freedom
f_2	: number of joints with two degrees of freedom
f_3	: number of joints with three degrees of freedom
f_4	: number of joints with four degrees of freedom
f_5	: number of joints with five degrees of freedom
b_1, b_2,.., b_6	: vertices of the base platform of the mechanism
P_1, P_2, P_3	: vertices of top platform of the mechanism with respect to the coordinate system at the base platform
p_1, p_2, p_3	: vertices of top platform of the mechanism with respect to the coordinate system at the top platform
L_a, L_b, L_c	: side lengths of the base platform
L_{p1}, L_{p2}, L_{p3}	: side lengths of the top platform
L_1, L_2,..., L_6	: leg lengths
Q_1, Q_2, Q_3	: projections of the vertices P_1, P_2, P_3 on to the sides L_a, L_b and L_c
L_{b1}	: distance between b_1 and Q_1
L_{b2}	: distance between b_2 and Q_1
L_{b3}	: distance between b_3 and Q_2
L_{b4}	: distance between b_4 and Q_2
L_{b5}	: distance between b_5 and Q_3
L_{b6}	: distance between b_6 and Q_3
m_1, m_2, m_3	: distances between points Q_i and P_i (i = 1, 2, 3)
β_1, β_2, β_3	: angles between the normals to the sides of the base platform and the x axis
w_1, w_2, w_3	: unit vectors in the Q_1P_1, Q_2P_2 and Q_3P_3 directions
S_1, S_2,.., S_6	: leg length vectors (i = 1, 2, .., 6)
R	: rotation matrix
R_x, R_y, R_z	: rotation matrices about the x, y, and z axes
x, y, z	: position of the center of gravity of the top platform
γ, β, α	: rotation angles about the x, y, and z axes
t	: translation vector
n_input	: number of neurons in the input layer
n_first_hidden	: number of neurons in the first hidden layer
n_second_hidden	: number of neurons in the second hidden layer
n_output	: number of neurons in the output layer
weight1	: first hidden layer weight matrix
weight2	: second hidden layer weight matrix
weight3	: output layer weight matrix
bias1	: first hidden layer bias
bias2	: second hidden layer bias

bias3	: output layer bias
input	: input data pattern
hidden1	: activation on the first hidden layer
hidden1_log	: output of the first hidden layer
hidden2	: activation on the second hidden layer
hidden2_log	: output of the second hidden layer
hidden3	: activation on the output layer
hidden3_log	: output of the neural network
d	: output layer error
e	: second hidden layer error
f	: first hidden layer error
target	: target data pattern
result	: output produced by the neural network
F	: activation function
$\mathbf{F^1(output)}$	: derivative of the activation function in the output layer
$\mathbf{F^1(hidden2)}$	: derivative of the activation function in the second hidden layer
$\mathbf{F^1(hidden1)}$	: derivative of the activation function in the first hidden layer
η	: learning rate
θ	: momentum factor
$\Delta\mathbf{weight1}_{previous}$	: previous amount of change in the first hidden layer weight matrix
$\Delta\mathbf{weight2}_{previous}$	: previous amount of change in the second hidden layer weight matrix
$\Delta\mathbf{weight3}_{previous}$	: previous amount of change in the output layer weight matrix
$\Delta\mathbf{bias1}_{previous}$	: previous amount of change in the first hidden layer bias
$\Delta\mathbf{bias2}_{previous}$	: previous amount of change in the second hidden layer bias
$\Delta\mathbf{bias3}_{previous}$	: previous amount of change in the output layer bias
bisector1	: angle bisector belonging to vertex P_1
bisector2	: angle bisector belonging to vertex P_2
bisector3	: angle bisector belonging to vertex P_3
vertex1,..,vertex3	: vertices of the top platform
cg	: coordinates of the center of gravity of the top platform
length1,..,length6	: lengths of the legs
λ_{ij}	: elements of the Eulerian matrix of rotation (i = 1, 2, 3, j = 1, 2, 3)
$\lambda_1, \lambda_2, \lambda_3$	: first, second and third rotation matrices
$\lambda_{Resultant}$	: resultant rotation matrix
L	: input vector of leglengths
$\mathbf{L_i}$	: result of the inverse kinematics equations
$\Delta\mathbf{L}$	: difference between the real input leglenths and the lengths found by the inverse kinematics equations
n_loop	: number of loops

POSITIONAL KINEMATICS ANALYSIS OF THE 6-3 STEWART PLATFORM MECHANISM USING HEURISTIC ALGORITMS

SUMMARY

Stewart Platform Mechanism (SPM) is used extensively in various areas of engineering, including flight simulators, driving simulators, amusement parks, oil platforms, robot arms, CNC machine tools, surface polishing, trimming, shaping and assembling, mounting antennas, and some medical devices. Piloting trainees are trained on flight simulators before flying the actual aircraft in order to minimize training time and losses. It is crucial for the trainees to feel the translational and rotational motion accurately. A parallel mechanism provides the motion that the pilot would be exposed to when flying on a real aircraft. A modified version of the 6 degree of freedom parallel mechanism proposed by Stewart in 1965 is used for this purpose. A 6-3 SPM used in flight simulators is composed of a fixed hexagonal base platform and a moving triangular top platform, linked by six rigid prismatic legs. The legs are often attached to the base at six points by universal joints and to the top at three points by spherical joints. Actuators that change leg lengths are at the base. The architecture lets the the top platform of the SPM have three translational and three rotational, meaning a total of six degrees of freedom with respect to the fixed base.

In this thesis, a detailed review of the literature on parallel mechanisms is given. Forward and inverse kinematics analyses of the 6-3 SPM are performed. Neural networks are explained in detail and are applied to the forward kinematics problem of the purely rotational, purely translational and general motions of the 6-3 Stewart platform mechanism. The loop method is used to improve the performance of the neural network model.

EVRİMSEL ALGORİTMALAR KULLANILARAK 6-3 STEWART PLATFORM MEKANİZMASININ KONUMSAL KİNEMATİĞİNİN İNCELENMESİ

ÖZET

Stewart Platform Mekanizması (SPM), başta uçuş simülatörleri, sürüş simülatörleri, eğlence parkları, petrol platformları, robot kolları, CNC tezgahları, yüzey cilalama, kesme, ve şekillendirme, montaj işlemleri, anten yerleştirme ve tıp olmak üzere mühendisliğin pek çok sahasında kullanılmaktadır. Eğitim zaman ve kayıplarını en aza indirmek için, pilot adayları gerçek uçakla eğitime başlamadan önce simülatörlerde eğitim görmektedirler. Pilot adaylarının, gerçek uçaktayken maruz kalacakları ötelenme ve dönme hareketlerini hassas olarak algılamaları çok önemlidir. Gerçek uçakta maruz kalınan hareketi, bir paralel mekanizma sağlar. 1965 yılında Stewart'ın önerdiği paralel mekanizmanın ufak değişikliklere uğramış bir versiyonu bu iş için kullanılmaktadır. Uçuş simülatörlerinde kullanılan 6 – 3 SPM altı adet boyları değişebilen bacak ile birbirine bağlanmış altıgen biçimli sabit bir alt platform ve üçgen biçimli hareketli bir üst platformdan oluşmaktadır. Bacaklar çoğu zaman tabana üniversal mafsallarla, üst platforma ise küresel mafsallarla bağlıdır. Bacak boylarını değiştiren eyleyiciler ise tabandadır. Bu mimari, üst platformun tabana göre üçü ötelenme üçü de dönme ile ilgili olmak üzere toplam altı serbestlik derecesine sahip olmasına izin verir.

Bu tezde, paralel mekanizmalar hakkında geniş bir literatür özeti verilmiştir. 6 – 3 SPM'nin düz ve ters kinematiği verilmiştir. Yapay sinir ağları detaylı bir şekilde açıklanmış ve 6 – 3 Stewart platform mekanizmasının yalnızca dönme, yalnızca ötelenme ve genel hareketlerinin düz kinematik problemlerine uygulanmıştır. Yapay sinir ağının performansını yükseltmek için çevrimmetodu uygulanmıştır.

1. INTRODUCTION

1.1 General Overview and History

The purpose of using simulators while training pilots is to reduce training time and expense that would be present when training on an actual aircraft. Simulators should be able to make the trainee sense every motion that he would possibly be exposed to in real flight, such as translational motion, rotational motion, or the combination of both. Sensitivity and accuracy of flight simulators have increased tremendously, following the advances in technology. Simulators give pilots the chance to try critical emergency cases that can not be practiced safely on a real aircraft. For these reasons, flight simulators are used extensively while training pilots.

A parallel mechanism provides the translational and rotational motions that the pilot would be exposed to in real flight. Figure 1.1 shows a flight simulator and the parallel mechanism under it that is used to simulate the entire motion. A parallel mechanism is made of links connected to common platforms in parallel. Another common definition of a parallel mechanism is that parallel mechanisms are mechanisms in which the top platform is linked to the base by at least two independent kinematic joints. Since the links form closed loops, motion of the links of the chains is constrained.

Figure 1.1: A flight simulator

Stewart was the first to come up with the idea of exploiting a parallel mechanism as a flight simulator in 1965 [1]. Though it was a bit different than what is currently known as the Stewart platform mechanism, the mechanism had six motors, each linked to the base, and six degrees of freedom, three of which are translational degrees of freedom and three of which are rotational degrees of freedom. This mechanism consisted of a triangular upper platform, three prismatic legs connected to it via spherical joints and three other legs giving rotational motion to the legs connected to the triangular platform. (Figures 1.2 and 1.3.)

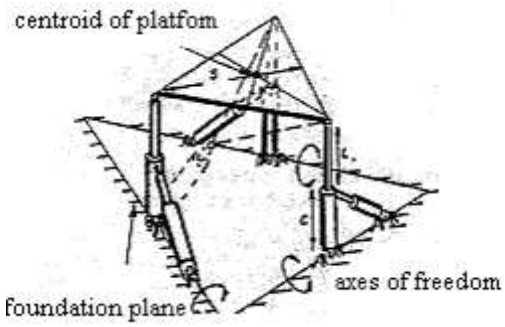


Figure 1.2: Architecture of the parallel mechanism proposed by Stewart [1]

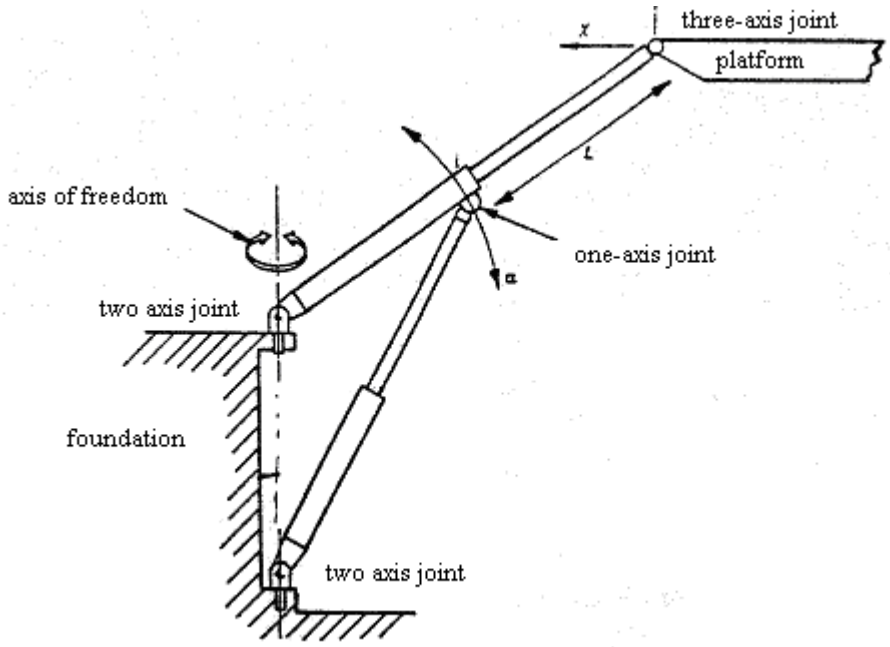


Figure 1.3: Architecture of a leg proposed by Stewart [1]

There have been numerous correspondances between Stewart and other researchers after this paper was published. V.E. Gough's contribution is the most important among all because he suggested having all six actuators operate in parallel. Thus, the mechanism attained the fully parallel character that it has today, and is sometimes referred to as the Gough-Stewart platform. In the following years, parallel mechanisms began to be used extensively in flight simulators, game simulators, oil platforms, robot manipulators, CNC machine tools, surface finishing, edge finishing, surface polishing, routing, profile milling, assembly, and medical applications.

Parallel mechanisms are robots in which a fixed platform and a moving platform are connected by links. Prismatic actuators in the links change the lengths of the links. Another definition is that parallel mechanisms are chains with one or more closed loops where only a few of the links are actuated. In parallel mechanisms, the end-effector is connected to the base by two or more in-parallel series links. In contrast to serial mechanisms, not all joint variables can be specified independently. A modified version of the mechanism developed by Stewart is the most popular kinematic chain of this type. This mechanism, known as the Stewart Platform Mechanism (SPM), has six controllable degrees of freedom. A typical SPM consists of a fixed rigid platform and a rigid moving platform linked by six prismatic legs, as shown in Figure 1.4. Prismatic legs that connect the vertices of the top and the base platform can simply be pistons, or they can be a series of elements linked by joints. Even though the base platform is fixed and the top platform is moving in flight simulators and most other applications, it is possible to have a fixed top platform and a moving base platform, as in the case of oil platforms. In the case of flight simulators, actuators change links lengths and move the upper platform. The upper platform has six degrees of freedom of motion with respect to the base platform (Figure 1.5). If the legs are assigned certain fixed lengths, the mechanism becomes a structure. Although it is sometimes referred to as the Gough-Stewart platform, this mechanism is most often referred to as the SPM since Stewart was the first to come up with the idea of using a parallel mechanism with three translational and three rotational degrees of freedom as a flight simulator.

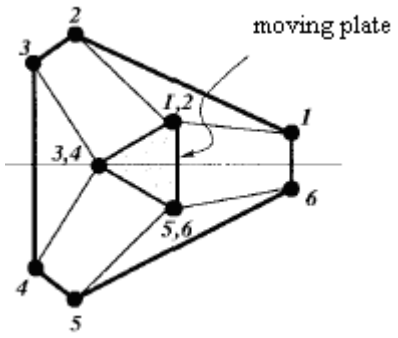


Figure 1.4: Top view of the SPM

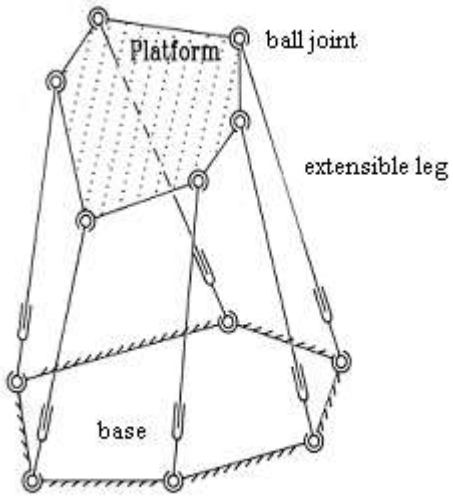


Figure 1.5: Side view of the SPM

1.2 Comparison with Serial Mechanisms

Traditional industrial open loop robots are andromorphic mechanisms with serially linked parts, bearing the advantages and disadvantages of the human arm (Figure 1.6). Although serially linked mechanisms have a higher reach and a larger workspace, it is observed that parallel mechanisms are superior when their dynamic characteristics are compared. Serial mechanisms are not as rigid as parallel mechanisms, as mentioned by various researchers [3, 4, 5, 6, 7, 8] and have low natural frequencies, leading to poor performances under high speeds and heavy loads. Additionally, each link, starting from the one linked to the ground until the tip of the manipulator, has to be large enough to carry the weights of the links and motors preceding it while retaining the desired accuracy. This causes an increase in the

dimensions and weight of the actuators, leading to a decrease in the dynamic performance of the robot in cases where high speeds and heavy loading are required. Another disadvantage of serially linked open loop manipulators is that actuator errors are summed, leading to a high error at the end effector of the robot. The same researchers have also given information on the superiority of parallel mechanisms over serial mechanisms. Parallel mechanisms are preferred in applications where dynamic loading is high, speed and accuracy are of greatest concern and workspace volume is of less importance, due to the fact that they have higher rigidity and accuracy. In contrast to serial manipulators, only some of the links are set into motion by the actuators. Moving parts of a parallel manipulator are not excessively heavy since the actuators are mounted to the base. Additionally, actuator errors do not add to one another and lead to a high error at the tip, due to the closed loop architecture. When all of these are taken into consideration, it is obvious that parallel mechanisms are suitable to be employed in flight simulators. However, parallel mechanisms have smaller workspace and have complicated design and kinematics and are more difficult to control.

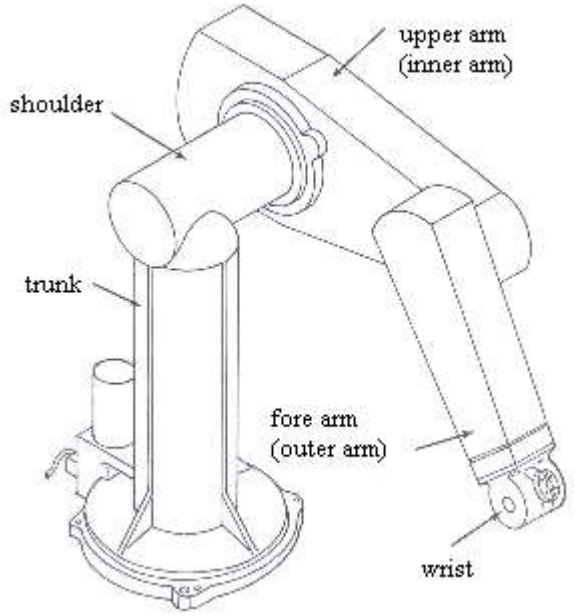


Figure 1.6: An anthropomorphic serial mechanism

Dasgupta and Mruthyunjaya [9] have stated that employment of parallel mechanisms in applications involving heavy loads and requiring high precision is present in biological life, as well. For instance, animals used as a means of carrying loads have more than two feet, resulting in higher stability than that of human beings. Other

examples are that humans use both of their arms in parallel when it is necessary to carry a heavy load and fingers function in parallel in handwriting, which is an operation requiring a high a level of precision. In addition to these, there exist mechanisms called hybrid mechanisms in which both parallel and serial mechanisms are used simultaneously or parallel mechanisms are combined in series. Figure 1.7 shows a hybrid manipulator. Some work conducted on their design and kinematic analysis can be found in literature [10, 11].

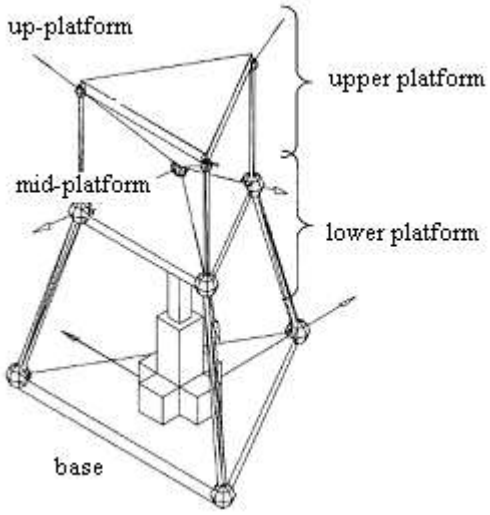


Figure 1.7: A hybrid manipulator

1.3 Literature Review

SPM has been analyzed to a great extent, due to its widespread usage in flight simulators. Although the legs are often attached to the base platform by universal joints and to the top platform by spherical joints, it is possible for the legs to be attached to both the base and the top platform by spherical joints. In the 6-3 SPM, the prismatic joints changing the lengths of the legs are active joints while all the other joints are passive joints. Nomenclature of the SPM is based the types and the number of joints used in the architecture. A mechanism whose links are connected to the base platform at m points and to the top platform at n points is called an m-n SPM. For example, if consecutive legs of a mechanism with six legs are attached together to form three groups of two and are attached to the top platform at three points, the mechanism is called a 6-3 SPM (Figure 1.4). This architecture is used quite commonly in robot mechanisms. There is one other special form of the SPM, called

the octahedral arrangement, where the legs are attached to both the upper and the base platform at 3 points in groups of two (Figure 1.8). An inspection on joint types reveals that U (or sometimes T) stands for universal joints, P stands for prismatic joints, R stands for revolute joints, C stands for cylindrical joints and S stands for spherical joints. A mechanism whose prismatic legs are attached to the base through universal joints and to the top platform through spherical joints is called a UPS SPM (Figure 1.9); while a mechanism whose legs are attached to both platforms through spherical joints is called a SPS mechanism. There exist a vast amount of work in literature on 3-3, 6-6, and 6-3 mechanisms and a small amount of work on 5-5, 4-4 and 6-5 SPMs. In addition, there are researchers who have designed new mechanisms in order to improve certain conditions related to the SPM [12]. Among all SPM types, 6-6 SPM is the most rigid one and has the best force distribution since it has six points of attachment both to the top and to the base platform.

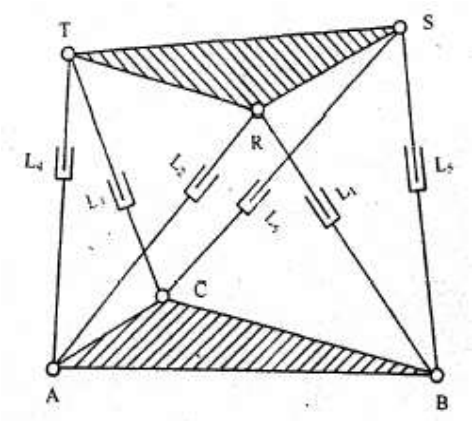


Figure 1.8: Octahedral SPM

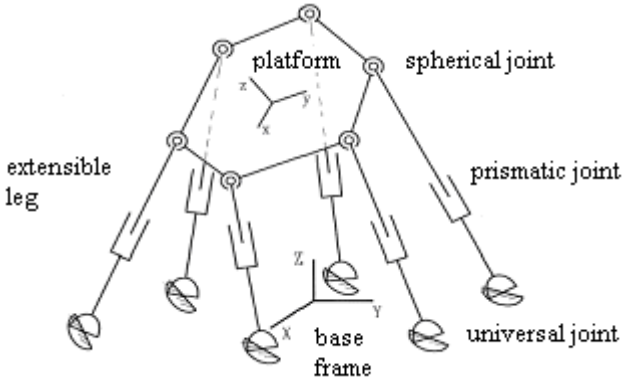


Figure 1.9: 6-6 UPS Stewart Platform mechanism

1.3.1 Parallel mechanism kinematics in literature

SPMs forward kinematics (FK) problem is defined as the problem of finding the translation and rotation of the upper platform with respect to the base platform when the lengths of the legs are known. In other words, the transformation of coordinates representing the position and orientation of the upper platform relative to the base platform need to be found when the lengths of the legs are given. FK problem is essential because all the configurations that the SPM can possibly have with a given set of leg lengths need to be known. Inverse kinematics problem (IK) is defined as the problem of finding the leg lengths that will produce a certain translation and rotation when that translation and rotation with respect to the base platform are known. The contrast between serial and parallel mechanisms is apparent here, as well. The FK problem of serial mechanisms is straightforward while their IK problem is rather complicated, whereas in parallel mechanisms the IK problem is simple and the FK problem is intricate. IK needs to be solved on-line for real time trajectory tracking. By inspecting the works of various researchers, it can be stated that the solution of the FK problem is not unique in the case of parallel mechanisms. This means that there exists more than one possible configuration for a set of given leg lengths. The solution of the IK problem, on the other hand, is unique, meaning only one set of leg lengths can produce a certain position of the upper platform.

The kinematics problem needs to be solved accurately for real-time control of the mechanism. Innocenti and Parenti-Castelli [8] have stated that one other advantage of having a solution to the FK is that the effect of input errors on the position of the moving platform of the SPM will be known accurately so that high-precision positioning of the moving platform will be possible.

Lee and Shah [6] have done the kinematics analysis of a 3 degree-of-freedom parallel manipulator shown in Figure 1.10. The proposed tripod mechanism has two degrees of orientational freedom and one degree of translational freedom. Such 3 DOF parallel mechanisms may be used to form 6 DOF parallel mechanisms by connecting two 3 DOF mechanisms in series.

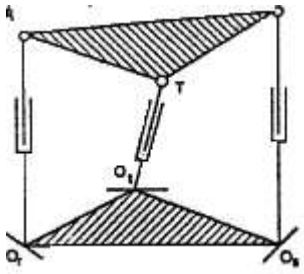


Figure 1.10: Triple arm mechanism

Dasgupta and Mruthyunjaya [15] have brought a canonical solution to the FK problem of the 6-6 SPM, revealing that the mechanism can have at most 64 assembly configurations for a set of given leg lengths. The same researchers have also proposed a predictor-corrector algorithm to solve the FK problem of the SPM, so that one does not have to deal with deriving the kinematics equations [16].

Nanua et al. [13] have solved the FK problem of the 6-3 SPM. They have derived the kinematics relations for both the 6-3 and the octahedral arrangement of the SPM. The FK problem is expressed as a polynomial of order 16, which means that there are 16 different configurations that the top platform of the SPM can have for a set of leg lengths.

Innocenti and Parenti-Castelli [8] have solved the FK problem of a general model of the SPM in closed form. Once again, the researchers have reached a polynomial of order 16, verifying that the top platform of the SPM can be assembled in 16 different configurations for a set of leg lengths. It must be noted here that not all of the 16 possible configurations are physically feasible. When mechanical constraints such as limitations on leg lengths, limitations on passive joint angles and the problem of interference of legs are taken into consideration, some of those 16 results have to be eliminated [82].

Shi and Fenton [7] have presented a method for solving the instantaneous FK of the general SPM, which can be applied to any 6 DOF parallel manipulator. Instantaneous FK problem is the problem of computing the generalized velocity of the end effector when the velocity vectors of the active joints are given. It is shown that the solution can be obtained by solving a set of six linear equations for the general case.

Merlet [3] has solved the FK of parallel manipulators with four different numerical methods and compared them on the grounds of computation time, while Liu et al. [5]

have solved SPMs FK by solving a set of three simultaneous non-linear algebraic equations and repeated that only a part of the solutions are feasible due to mechanical constraints.

Sreenivasan et al. [79] have investigated the FK problem of the 6-6 SPM and reached a set of decoupled polynomial equations. Once again, it is shown that the SPM can be assembled in sixteen different configurations for a given set of leg lengths. Wohlhart [14] has also worked on the FK of a special configuration of the SPM, the spherical SPM, shown in Figure 1.11. The spherical SPM is another six-legged mechanism where three actuators act on the platform through the use of a single spherical joint and control the position of one point of the platform while the other three legs are responsible from the orientation of the platform by gyrating it about this point. The author has shown that 16 configurations of the top platform are possible for a set of leg lengths, though not all of them are real and not all of them are physically feasible.

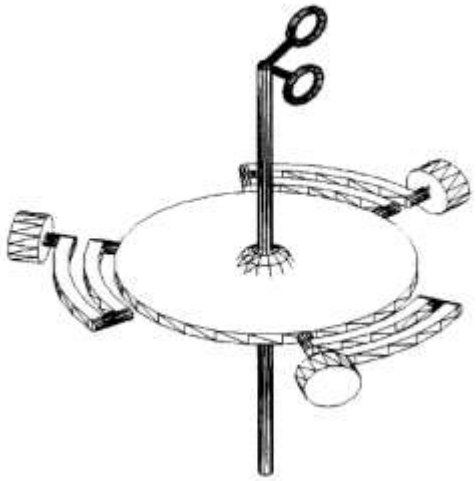


Figure 1.11: Spherical SPM

Ku [2] has solved the FK of the octahedral SPM (Figure 1.8.) via a simple and cost-effective numerical algorithm based on Newton-Raphson method. Though the same mechanism is examined and thus the same FK equations are derived in [13] and in [2], [2] yields results only in a few iteration steps. Zhao and Peng [17] have also solved the FK of parallel mechanisms via a new numerical method. Jakobovic and Jelenkovic [18] have stated that the FK problem of parallel mechanisms can be solved using optimization algorithms.

Works on some special configurations of the SPM also exist in literature. Tsai et al. [19] have solved the FK problem of the 3-PRS (prismatic-revolute-spherical) parallel mechanism (Figure 1.12). Kim and Park [20] have solved the FK problem of the 3-RS parallel mechanism proposed to overcome the small workspace problem of the 6-6 SPM. In the 3-RS platform, the base platform and the moving top platform are linked by three serial links. All of these links include a passive spherical and a passive revolute joint (Figure 1.13). Predictably, the FK problem results in a polynomial of order 16. Di Gregorio [21] has solved the FK problem of the spherical mechanism with revolute actuators on the base and passive cylindrical joints that he calls the 3-URC (Figure 1.14) wrist. Callegari and Tarantini [22] have solved the FK and IK problems of the 3-PRC parallel mechanism that allows only purely translational motion in the presence of certain constraints (Figure 1.12). Di Gregorio [23] has made proposals on the architecture of purely translational parallel mechanisms, but such mechanisms are not useful in aerospace applications since they have only three degrees of freedom of motion.

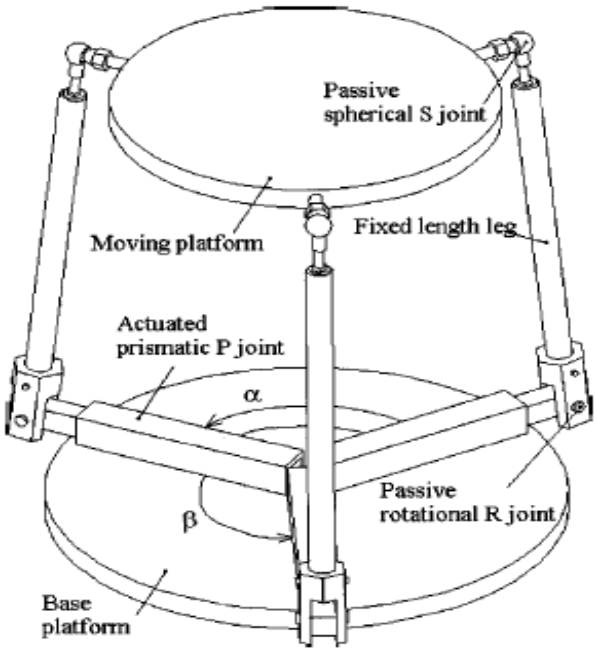


Figure 1.12: 3-PRS mechanism

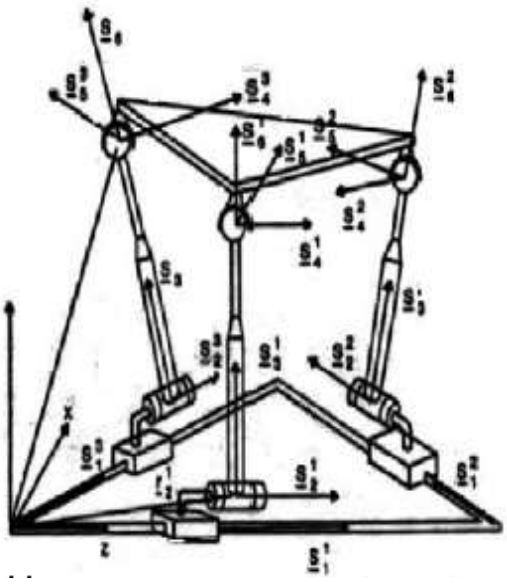


Figure 1.13: 3-RS mechanism

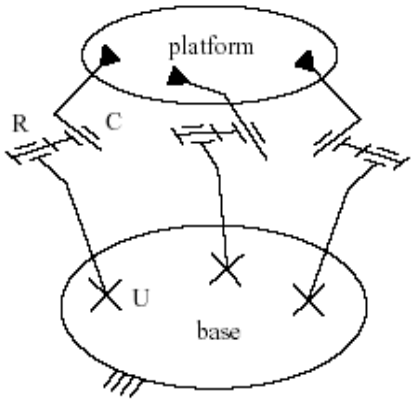


Figure 1.14: 3-URC mechanism

1.3.2 Parallel mechanism dynamics in literature

Dynamics of parallel mechanisms examines the relation between the input forces and moments and the motion of the end effector. The human body is more sensitive to changes in the magnitude and direction of velocity, namely acceleration, than the magnitude of the velocity. Therefore, it is utterly important for accelerations to be calculated correctly. Forward dynamics solves for the acceleration of the moving platform when the positions, velocities and moments of the joints and the mass distribution of the system are given. Inverse dynamics solves for the joints force and moments that will produce a certain acceleration at the end effector when the desired acceleration of the moving platform, positions and velocities of the joints and the

mass distribution of the system are given. Dynamic modeling of simulators is an important issue because simulators are supposed to make piloting trainees feel the forces and moments he would be exposed to in real flight. The two main approaches in dynamic modeling of parallel mechanisms are the Newton-Euler method and Lagrange's method. In addition to these two methods, there are researchers who have employed virtual work method and Kane's dynamic equation in solving the dynamics of the SPM.

In the Newton-Euler approach, dynamics of all the parts composing the system are calculated separately and instantaneous or infinitesimally small aspects of motion are inspected, meaning that the Newton-Euler method is a differential method. Vectoral quantities, namely velocities and forces are used in the calculations. The fact that complicated differentiations are not necessary in this approach is a superiority when compared to Lagrange's method. A free body diagram must be drawn for each component of the system and Newton-Euler equations must be applied.

Dasgupta and Mruthyunjaya [24] have derived closed-form dynamics equations of the 6-UPS and 6-SPS SPMs by using Newton-Euler approach. It should be noted that this method includes all dynamic forces, namely inertial, centrifugal, coriolis, gravitational and viscous friction forces, thus the method yields a perfect model. The same researchers have also solved for the inverse dynamics of the SPM by using Newton-Euler approach [25]. Similar to [24], the model in [25] includes all dynamic forces in the model.

Lagrange's method is simpler to implement and encompasses the kinetic and potential energies of the entire system. Lagrangian dynamics examines the system in a finite interval of time, making the method an integral method. Scalar quantites are used in calculations because energy relations are applied. G. Lebret et al. [27] have solved the forward dynamics of the SPM with this approach. Lagrange's formula is applied after the kinetic and potential energies of the base and the top platform and the legs are found. Pang and Shahinpoor [28] have solved for the inverse dynamics of a 3 degree of freedom mechanism resembling the SPM using Lagrange's method. However, in this method many symbolic computations must be carried out during the differentiations.

Wang and Gosselin [26] have employed the method of virtual work in the dynamic analysis of the SPM and concluded that this method is efficient in terms of computation time. Liu et al. [29] have solved SPMs dynamics by making use of Kane's dynamic equation.

1.3.3 Singularities and workspace analysis in literature

Although parallel mechanisms are used extensively in many industrial and engineering applications, there is no generally accepted approach to determine their workspaces due to the complexity of their configurations and kinematics analyses. The workspace of a parallel mechanism has to be computed during the design phase. It can be described as the region reachable by the center of gravity of the moving platform, under constraints on active and passive joints and leg lengths. Space reachable by the SPM is dependant upon mechanical and geometric constraints due to its architecture which are due either to the limits on the lengths of the legs, or to the limits on the mobility of the joints, or to the interference of the legs. [30]. The reason why there is no general approach to determine the workspace of a parallel mechanism is that FK equations need to be used during the computation of the workspace and those equations, as stated above, are fairly complicated.

Literature refers to more than a single type of workspace. For instance, reachable workspace is described as the set of points reachable by the center of gravity of the moving platform of the SPM, while orientational workspace is described as the set of rotations that the moving platform can undergo while its center of gravity is fixed to a point in space [31]. Previous work on the workspace analysis of parallel mechanisms can be divided into two main groups. These are translational workspace analysis and orientational workspace analysis. Translational workspace analysis determines the set of feasible translations of the moving platform in the x, y, and z directions while the platform has a constant orientation. Orientational workspace analysis determines the set of feasible rotations of the moving platform about the x, y, and z axes while its center of gravity is fixed to a point in space. Most of the recent studies are on orientational workspace analysis [32]. The architecture of the SPM affects the size, shape and symmetry of the workspace because the transformation matrix relating the velocities of the legs to the velocity of the moving platform is dependant on it [33]. The Jacobian matrix relating the linear and angular velocities of

the legs to those of the moving platform is a 6*6 matrix in a mechanism with 6 degrees of freedom [34]. Another difficulty with determining the workspace of the SPM is that the 6 dimensional workspace of the 6 degree of freedom mechanism can not be represented graphically. It can be represented graphically only as smaller dimensional subsets of the 6 dimensional workspace. In other words, out of the six degrees of freedom of the mechanism, three must be held constant and the effect of the other three on the workspace can be shown on a graph [31].

One of the most important problems encountered during the analysis, design, motion planning and control of parallel mechanisms is the problem of singularities. Due to the closed loop structures of parallel mechanisms, their motion is restricted and singularities occur in the workspace. As stated above, a closed loop structure is a mechanism where the links form at least one closed loop. Singularities are points where either the forward or the inverse kinematics problem has no solution. The rank of the Jacobian matrix relating the velocities of the legs to the velocities of the moving platform drops at singular loci [35]. Determination of singular locations by using the Jacobian matrix method is difficult because the determination of the inverse of the Jacobian matrix is complicated [36], but Collins and McCarthy [37] have found the singularities of a spatial parallel mechanism with triangular base and moving platform with the Jacobian transformation matrix method. It is possible to classify singularities by examining the properties of the Jacobian matrix. Angeles et al. [38] have found the singularities of a tripod 6 degree of freedom parallel mechanism with two actuators on each of its legs by examining the Jacobian matrix, as well.

Another reason to determine the singular configurations of parallel mechanisms is that such a mechanism gains an uncontrollable degree of freedom at these points, thus such positions must be avoided. Singular configurations of a parallel mechanism must be determined for the design, control, determination of the singularity free workspace of the mechanism and singularity free path planning. Kim and Chung [39] have done an analytical study to determine the singular configurations of the general SPM. Bessala et al. [40] have stated that even though the workspaces of planar parallel mechanisms can be determined analytically, determination of the workspace of a spatial parallel mechanism is tedious. For this reason, most of the studies on the determination of the workspaces and singular locations of the SPM incorporate

numerical analysis [41]. D. Zlatanov et al. [42,43] have defined and classified the singularities of a general mechanism by using information on instantaneous kinematics. Huang et al. [44] have done the singularity analysis of the SPM by using rigid body kinematics.

1.3.4 Limited degree of freedom mechanisms in literature

Mechanisms with less than 6 degrees of freedom, often referred to as limited degree of freedom mechanisms, are advantageous in applications where production rate and cost is the main concern because their assembly is simpler, costs are lower, control algorithms are simpler and speed capacities are higher, as mentioned by Joshi and Tsai [34]. They have received quite a lot of attention although they can not be used as flight simulators. These researchers have found a method to determine the singular loci of mechanisms with limited degrees of freedom, which depends on determining the Jacobian matrices of such systems. An inspection of the Jacobian matrix also reveals the type of the singularity.

Carretero et al. [33] have determined the workspace of a 3 degree of freedom parallel mechanism. Even though this mechanism can not be used in aerospace applications such as in a flight simulator, it can be used in applications that do not require 6 degrees of freedom. In such cases, the use of limited degree of mechanisms reduces costs. It is shown that changing architectural parameters changes the size of the workspace. Du Plessis and Snyman [45] have found the workspace of the 6-3 SPM by using numerical methods.

Romdhane et al. [41] have examined a mechanism with 3 translational degrees of freedom and found singular locations through a kinematic analysis. D. Gregorio and Parenti-Castelli [30,46] have found the reachable workspaces and singularities of the 3-UPU and U-2PUS mechanisms with 3 translational degrees of freedom analytically. 6 degree of freedom mechanisms which can move in 3 translational and 3 rotational directions in space can be comprised of a 3 degree of freedom purely translational mechanism and a 3 degree of freedom purely rotational mechanism linked in series. Thus, when workspaces are examined, purely translational or purely rotational mechanisms should also be examined. Yang et al. [47] have done the singularity analysis of tripod parallel mechanisms by using passive joint velocities. Liu et al. [48] have found the singularities of a 3 degree of freedom mechanism with

revolute actuators called the HALF parallel mechanism. Limited degree of freedom mechanisms are used extensively in industrial applications since their singularities can be found effortlessly, though they can not be used in flight simulation applications.

1.3.5 Workspace and singularities of 6 degree of freedom mechanisms in literature

Liu et al. [49] have found the singularities of parallel mechanisms by making use of geometrical constraint equations in differential form. Kim et al. [50] have computed the workspace of a 6 degree of freedom parallel mechanism through a geometric approach. Geometric approach implies that the information needed to compute the solution is contained within the geometry of the mechanism, such as the dimensions of the platform, positions of the joints and leg lengths. The advantage of such a method is that merely design parameters are enough in order to compute the workspace so that information on position and orientation of the platform are not needed. Similarly, Wen and O'Brian [51] have found the singularities of a tripod spatial mechanism geometrically, without dealing with the coordinates of the moving platform. Pernkopf and Husty [36] have found the singularities of the SPM geometrically, while Wolf and Shoham [52] found the singularities of parallel mechanisms geometrically, as well. Moving the platform on a singularity free path is only possible after the singularities are found.

Wang and Min [53] have computed the limits of the workspace of the 6-3 SPM for the case where the moving platform is parallel to the base. Wang et al. [54] have computed the limits of the workspace of a 6 degree of freedom mechanism. Angeles et al. [55] have done the singularity analysis of the 6 degree of freedom RRRS mechanism with two actuators on each one of its three legs. Additionally, they have proposed using modular structures in parallel mechanisms in order to overcome singularities. Majid et al. [56] have done the singularity and workspace analyses of the 6 degree of freedom 3-PPSR mechanism which is simpler and more rigid when compared to the general SPM. Merlet [57] has classified the methods of finding the workspaces of parallel manipulators as the discretization method, geometric method and the Jacobian matrix method, and has worked on the orientation workspace of the general SPM under three types of constraints. These mechanical constraints are limitations on leg lengths, limitations on passive joint angles and interference of the

legs. In the discretization method, the end effector of the mechanism is moved incrementally and the IK equations are solved to obtain link lengths. These links lengths are then checked to see whether they satisfy the mechanical constraints listed above. This method is not computationally efficient especially when it is desired to obtain a high resolution map of the workspace. In the geometric approach, mechanical constraints are omitted and it is accepted that the boundary of the workspace is reached when at least one of the actuators reach its limits. This method should be used when it is desired to determine the physical limits of the workspace.

Badescu et al. [58,59] has done the workspace optimization of the purely translational 3-UPU and purely rotational 3-UPS mechanisms. Since it is proposed to use modular structures while constructing parallel mechanisms, it is necessary to examine the workspaces of purely translational or purely rotational mechanisms as they may be used as part of 6 degree of freedom mechanisms. Bonev and Ryu [60] have found the 3 dimensional orientation workspace of 6 degree of freedom mechanisms using the discretization method.

1.3.6 Proposals in literature

Merlet [62] has worked on the optimization problem of the SPM and formulated the problem so that there are 48 optimization parameters related to the architecture of the mechanism. These parameters are the coordinates of the vertices of the top and the base platform, leg lengths and changes in leg lengths. The objective of this optimization procedure was to maximize workspace, accuracy, rigidity, and the forces on the moving platform for a given actuator force. Monsarrat and Gosselin [63] have proposed a new 6 degree of freedom tripod design with five links in each leg (Figure 1.15.) This mechanism is statically balanced meaning no actuator forces or moments are required when the mechanism is in rest. The researchers have first maximized the singularity free translational workspace of the mechanism and then maximized the 6 dimensional workspace of the mechanism using the discretization method since it may be used in applications such as flight simulators.

Jin et al. [64] have proposed a selectively actuated tripod parallel mechanism where the moving platform can make either 3 dimensional or 6 dimensional motion in space, depending on whether the actuators make linear or angular motion.

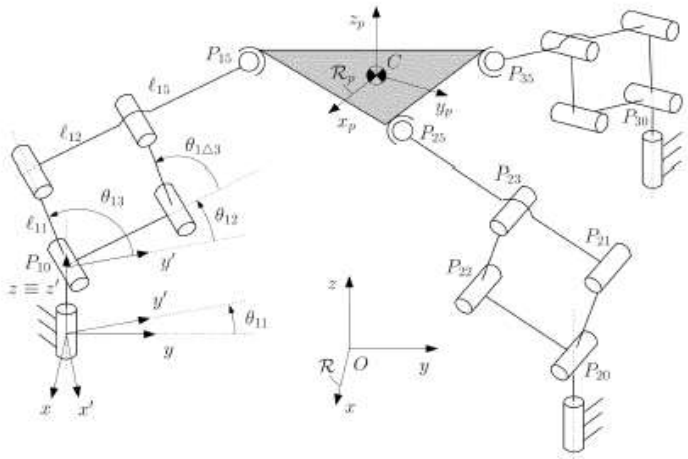


Figure 1.15: A parallel mechanism design proposal

Another design proposal aiming at maximizing the workspace of spatial parallel mechanisms is done by Lee et al. [65] where the actuators change the distance between the center of the moving platform and the point where the legs are attached to the moving platform (Figure 1.16). The workspace of such a SPM is 1.5 times greater than that of a classical SPM.

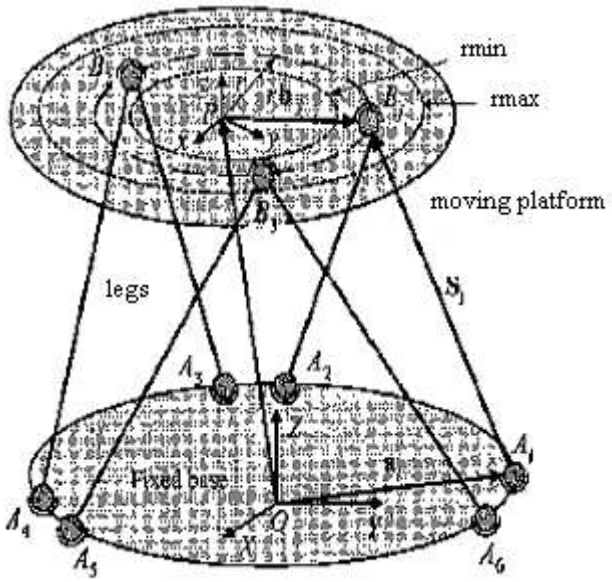


Figure 1.16: An SPM design proposal

Chen et al. [66] have proposed joint coupling to overcome singular loci meaning one actuator can set more than one joint into motion at a time. Thus, it is possible to change singularity loci without changing the architecture of the mechanism. Arai et al. [67] have proposed magnifying the workspace by changing link parameters. The

idea here is to change the link parameters of a single mechanism and obtain mechanisms with small workspaces covering a huge workspace and thus envelop a workspace larger than that of the single mechanism with singularities. Although this method can be applied to 6 degree of freedom micromanipulator mechanisms, it is not applicable to flight simulators. Miller [68] has studied the effect of motor axis directions on the shape and volume of the workspace.

1.3.7. Path planning in literature

Path planning is not a commonly addressed issue in the area of parallel mechanisms. Cortes and Simeon [70] have proposed a method based on probabilistic motion planning. Merlet [71] has worked on an algorithm which finds whether or not a straight line connecting two points, meaning two different positions of the SPM, lies completely within the workspace or not. Dash et al. [72] have proposed a method for singularity free path planning of parallel mechanisms. In this case, the singularities within the workspace are considered to be obstacles and thus the motion is planned without moving near those singularities. The researchers have mentioned that this method is applicable to any mechanism made of revolute and prismatic joints.

1.3.8 Application of heuristic algorithms in literature

Heuristic algorithms such as neural networks and genetic algorithms may be employed in solving the kinematics, dynamics, singularity and workspace problems of the SPM. Genetic algorithms (GA's) are search and optimization methods based on the principle of natural selection and always converge to a possible solution. Probable solutions that compete with each other are expressed in terms of various parameters and form a population. These potential solutions called chromosomes are data strings and their closeness to the solution is tested by a previously determined fitness function. Variety of the chromosomes in the society increases through selection, cross over and mutation, and thus evolutionary progress takes place, similar to the process in biological systems in nature. Evolutionary progress of the chromosomes, meaning the possible solutions, provides convergence to the real solution. Toogood et al. have done motion planning for a 3 degree of freedom revolute mechanism in an environment with obstacles using GA's. Even though GA's always converge to a solution, their disadvantage is that their computation time is not suitable for real time applications [73]. Su et al. [74, 75, 76] have done the

optimization of the kinematic parameters and singularity analysis of a 6 degree of freedom SPM used in fine-tuning a telescope using GA's. Thus, the problem of finding singular locations turns into an optimization problem.

Artificial neural networks (NN's) are networks made of elements working in parallel, inspired by the neural networks in nature. The network function is determined by the connection weight values between processing elements called neurons, and biases. A NN is trained to produce certain outputs for certain inputs. It is possible for a NN to realize a function by changing its weights and biases. Yee and Lim [77] have solved the FK problem of the SPM by using NN's. Durali and Shameli [78] have solved the FK problem of the 6-6 SPM using NN's and used numerical methods to improve the NN output. This method is advantageous in terms of computation time such that once the NN is trained, the time required to obtain an output for an input set of leg lengths in is the order of milliseconds.

1.4 Organization of the Thesis

The remaining chapters of this thesis are organized as follows: In Chapter 2, it is shown that the mobility of the system is 6, and forward and inverse kinematics analysis of the 6-3 SPM is carried out. In Chapter 3, neural networks are explained in detail by stating their advantages, giving an explanation on their biological counterparts and details on the structures of artificial neural networks and artificial neurons, describing activation functions, and finally deriving the mathematical relations of the backpropagation algorithm. In Chapter 4, neural networks are applied to the forward kinematics problem of the 6-3 Stewart platform mechanism, architecture of the SPM used is given, the process of generating purely rotational, purely translational and general spatial data, where translation and rotation are combined, is explained, and the loop method which is used to improve the performance of the neural network is explained. In Chapter 5, results of the neural network training and applying the loop method are given. In Chapter 6, conclusions of this thesis are summarized.

2. KINEMATIC ANALYSIS

It was stated in the previous chapter that the forward kinematics problem of the Stewart Platform Mechanism is defined as the problem of finding the position of the center of gravity of the top platform of the SPM and the orientation of the top platform of the SPM with respect to the axes of the coordinate system fixed to the base, when the lengths of the six legs are given. Inverse kinematics problem is the problem of finding the leg lengths that will give a certain position and orientation of the top platform. Yurt [82] and Nanua et al. [13] have derived the 16^{th} order polynomial equation governing the FK of the SPM. IK analysis of the SPM, on the other hand, is straightforward.

2.1 Mobility of the System

Mechanisms are formed by links connected by joints. It is crucial to examine joints in a mechanism since joints permit or restrict motion in certain directions. The degree of freedom (dof) of a joint is the number of independent variables needed to describe the position of one of the elements it is linking, with respect to the other. Figure 2.1 gives the degrees of freedom of the six most common joint types. Likewise, the degree of freedom of a system made of rigid bodies is the minimum number of coordinates needed to determine the positions of all the bodies making up the system. The degree of freedom of a system can also be called its mobility. The mobility of a mechanism can be defined as the minimum number of coordinates needed to determine the positions of all the parts of the system with respect to a reference system. The mobility of a mechanism can be calculated as

$$M = 6(n-1) - 5f_1 - 4f_2 - 3f_3 - 2f_4 - f_5 \qquad \textbf{(2.1)}$$

where n is the number of links making up the system,

f_1 is the number of joints with dof = 1,

f_2 is the number of joints with dof = 2,

f_3 is the number of joints with dof = 3,

f_4 is the number of joints with dof = 4, and

f_5 is the number of joints with dof = 5.

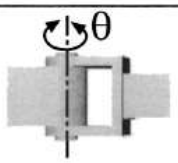
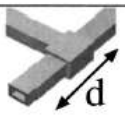
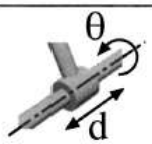
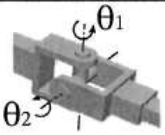
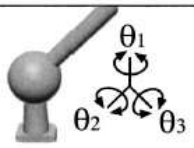
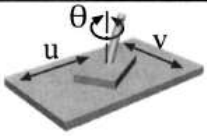
Joint	Diagram	Symbol	DOF
Revolute		R	1
Prismatic		P	1
Cylindric		C	2
Universal		T	2
Spherical		S	3
Planar		E	3

Figure 2.1: Six basic joint types [87]

In the case of the 6-3 SPM, there is one top platform, one base platform, and each leg is made of two links connected by a prismatic joint. This makes a total of 14 links. Legs are connected in groups of two (Figure 2.2) and linked to one another by revolute joints with one degree of freedom. This means that there are 3 joints in the system with one degree of freedom. Legs are connected to the base platform by universal joints and the upper and lower parts of the legs are connected to each other by prismatic joints which allow both translation in the direction of the legs and rotation about the leg. This means that there are 12 joints in the system with two degrees of freedom. Legs are connected to the top platform at three points by

spherical joints. This means that there are 3 joints in the system with three degrees of freedom.

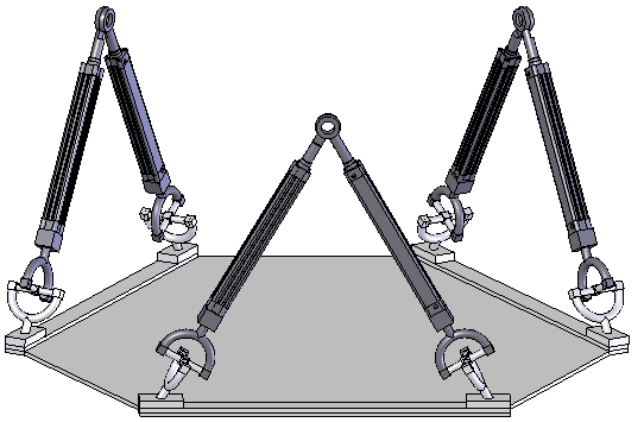


Figure 2.2: Points of attachment of the legs to each other in groups of two

Thus, in this system n = 14 **(2.2)**

$$f_1 = 3 \quad \textbf{(2.3)}$$

$$f_2 = 12 \quad \textbf{(2.4)}$$

$$f_3 = 3 \quad \textbf{(2.5)}$$

and the mobility of the system becomes

$$M = 6(14 - 1) - 5(3) - 4(12) - 3(3) \quad \textbf{(2.6)}$$

$$= 6 \quad \textbf{(2.7)}$$

This means that the motion of the mechanism can be described with respect to a reference system by six independent variables. This is true since the motion of the top platform can be described by six coordinates. Three of these are related to translation and the other three are related to rotation. The coordinates of the center of gravity of the top platform with respect to the reference system fixed to the base (x, y, z) and the angles of rotation of the top platform with respect to the three reference axes on the base (γ, β, α), meaning a total of six coordinates are sufficient to describe the exact state of the mechanism.

2.2 Forward Kinematics Analysis

Forward kinematics problem of the Stewart Platform Mechanism is defined as the problem of finding the position of the center of gravity of the top platform of the

SPM and the orientation of the top platform of the SPM with respect to the axes of the coordinate system fixed to the base, when the lengths of the six legs are given.

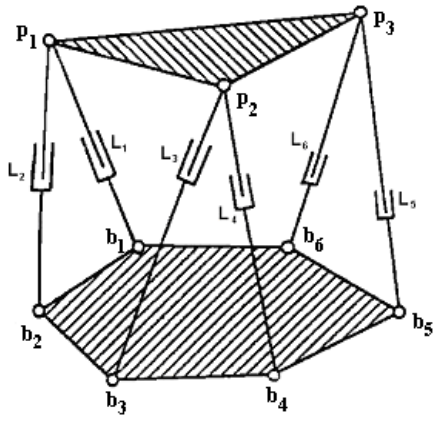


Figure 2.3: 6-3 SPM

Figure 2.3 shows a 6-3 SPM. The points where the legs are attached to the base platform are called b_1, b_2, b_3, b_4, b_5, and b_6, respectively, and the points where the legs are attached to the top platform are called P_1, P_2, and P_3. Side lengths of the base platform b_1b_2, b_3b_4, and b_5b_6 are called L_a, L_b, and L_c, respectively, and the side lengths of the top platform, P_1P_2, P_2P_3, and P_1P_3 are called L_{p3}, L_{p1}, and L_{p2}, respectively. Leg lengths are called L_1, L_2, L_3, L_4, L_5, and L_6, respectively. When triangle $b_1P_1b_2$ is examined by inspecting Figure 2.4, it can be observed that point P_1 moves on the circle with its center at O_1 having radius is m_1. This circle is the intersection region of two spherical surfaces. The first one of these surfaces has its center at b_1 and has radius L_1 and the second one of these surfaces has its center at b_2 and has radius L_2.

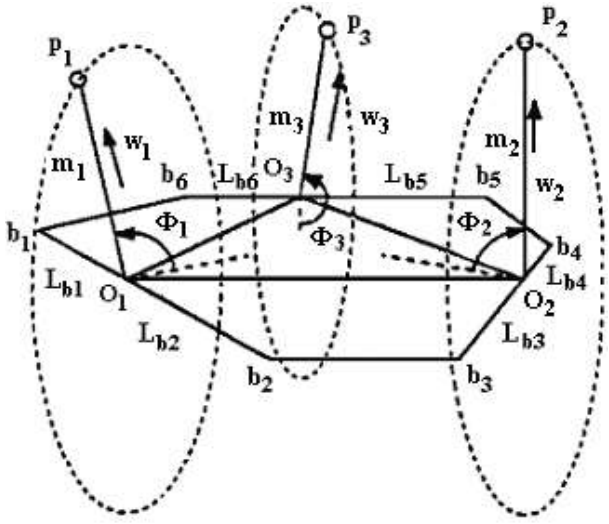


Figure 2.4: Variables and vectors used in calculations

Likewise, point P_2 is constrained to move on the circle with its center at O_2 having radius L_2 and points P_3 is constrained to move on the circle with its center at O_3 having radius L_3 [82, 13]. Thus, the mechanism can be represented by the equivalent mechanism given in Figure 2.5.

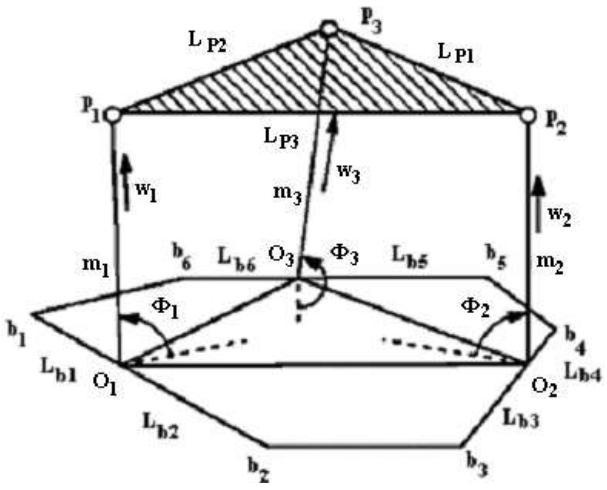


Figure 2.5: Model of the SPM used in the FK analysis

It can be further observed from Figure 2.5 that the distances between points O_1 and b_1 is L_{b1}, O_1 and b_2 is L_{b2}, O_2 and b_3 is L_{b3}, O_2 and b_4 is L_{b4}, O_3 and b_5 is L_{b5}, and O_3 and b_6 is L_{b6}.

Forward kinematics analysis starts by expressing the side lengths of the base in terms of the leg lengths. The triangles in Figure 2.6 show the triangular surfaces formed by the vertices of the top platform of the 6-3 SPM and vertices of the base platform of the 6-3 SPM.

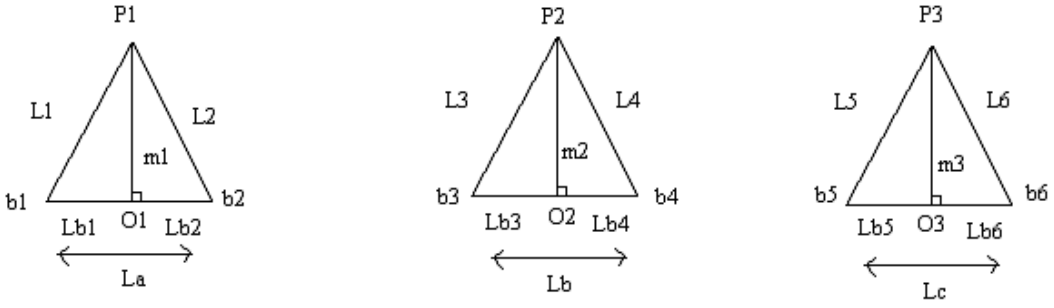


Figure 2.6: Triangular surfaces formed by the vertices of the top platform of the 6-3 SPM and vertices of the base platform of the SPM, positions of O_1, O_2, and O_3 and distances m_1, m_2, and m_3

In the figures above, P_1, P_2, and P_3, denote the vertices of the moving platform, b_1, b_2, b_3, b_4, b_5 and b_6 denote the vertices of the base platform, L_1, L_2, L_3, L_4, L_5 and L_6 denote the leg lengths of the 6-3 SPM, and La, L_b and L_c denote the side lengths of the hexagonal base platform. Distances between P_i's (i=1, 2, 3) and Q_i's (i=1, 2, 3) are called m_i such that $m_i = |P_i Q_i|$ (i=1, 2, 3). In this case, FK analysis may be carried out as follows.

$$L_a = L_{b1} + L_{b2} \quad \textbf{(2.8)}$$

When both sides are multiplied by $2L_{b1}$,

$$L_a\ 2L_{b1} = 2\ L_{b1}^2 + 2\ L_{b1}\ L_{b2} \quad \textbf{(2.9)}$$

$$L_a^2 = (L_{b1} + L_{b2})^2 = L_{b1}^2 + L_{b2}^2 + 2\ L_{b1}\ L_{b2} \quad \textbf{(2.10)}$$

$$2\ L_a\ L_{b1} = L_a^2 - L_{b2}^2 + L_{b1}^2 \quad \textbf{(2.11)}$$

From geometry,

$$m_1^2 = L_1^2 - L_{b1}^2 = L_2^2 - L_{b2}^2 \quad \textbf{(2.12)}$$

$$2\ L_a\ L_{b1} = L_a^2 - L_{b2}^2 + L_{b1}^2 + m_1^2 - m_1^2 \quad \textbf{(2.13)}$$

$$= L_a^2 + (m_1^2 + L_{b1}^2) - (m_1^2 + L_{b2}^2) \quad \textbf{(2.14)}$$

$$= L_a^2 + L_1^2 - L_2^2 \quad \textbf{(2.15)}$$

Thus,

$$L_{b1} = \frac{L_a^2 + L_3^2 - L_4^2}{2\ L_a} \qquad L_{b2} = L_a - L_{b1} \qquad m_1 = (L_1^2 - L_{b1}^2)^{1/2} \quad \textbf{(2.16)}$$

Likewise,

$$L_{b3} = \frac{L_b^2 + L_6^2 - L_5^2}{2L_b} \qquad L_{b4} = L_b - L_{b3} \qquad m_2 = (L_3^2 - L_{b3}^2)^{1/2} \quad \textbf{(2.17)}$$

$$L_{b5} = \frac{L_c^2 + L_5^2 - L_6^2}{2L_c} \qquad L_{b6} = L_c - L_{b5} \qquad m_3 = (L_5^2 - L_{b5}^2)^{1/2} \quad \textbf{(2.18)}$$

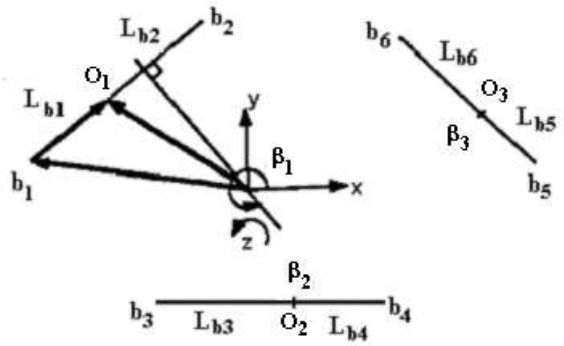


Figure 2.7: Description of the coordinate system and position vectors

The coordinate system where the position vectors belonging to O_1, O_2 and O_3 will be described has its xy plane coinciding with the base platform and its z axis is directed away from the base platform, as seen in Figure 2.7. The origin of the coordinate system fixed to the base is at the center of the base. If $\mathbf{b_1}$, $\mathbf{b_2}$, $\mathbf{b_3}$, $\mathbf{b_4}$, $\mathbf{b_5}$ **and** $\mathbf{b_6}$ are the position vectors of points b_1, b_2, b_3, b_4, b_5 and b_6, respectively, then $\mathbf{O_1}$, $\mathbf{O_2}$ **and** $\mathbf{O_3}$ may be described as the following.

$$\mathbf{O_1} = \mathbf{b_1} + \frac{L_{b1}\,(\mathbf{b_2} - \mathbf{b_1})}{|\mathbf{b_2} - \mathbf{b_1}|} \qquad \textbf{(2.19)}$$

$$\mathbf{O_2} = \mathbf{b_3} + \frac{L_{b3}\,(\mathbf{b_4} - \mathbf{b_3})}{|\mathbf{b_4} - \mathbf{b_3}|} \qquad \textbf{(2.20)}$$

$$\mathbf{O_3} = \mathbf{b_5} + \frac{L_{b5}\,(\mathbf{b_6} - \mathbf{b_5})}{|\mathbf{b_6} - \mathbf{b_5}|} \qquad \textbf{(2.21)}$$

In a more compact form,

$$\mathbf{O_i} = \mathbf{b_{2i-1}} + \frac{L_{b(2i-1)}\,(\mathbf{b_{2i}} - \mathbf{b_{2i-1}})}{|\mathbf{b_{2i}} - \mathbf{b_{2i-1}}|} \qquad (i=1,2,3) \qquad \textbf{(2.22)}$$

Angles β_1, β_2, and β_3 are the angles between the normals to the sides of the base platform and the x axis and can be expressed as the following.

$$\beta_1 = \sin^{-1}\frac{\{(\mathbf{b_2} - \mathbf{b_1}) \times \mathbf{k}\} \times \mathbf{i}}{|(\mathbf{b_2} - \mathbf{b_1}) \times \mathbf{k}|} \qquad \textbf{(2.23)}$$

$$\beta_2 = \sin^{-1} \frac{\{(\mathbf{b}_4 - \mathbf{b}_3) \times \mathbf{k}\} \times \mathbf{i}}{|(\mathbf{b}_4 - \mathbf{b}_3) \times \mathbf{k}|} \quad (2.24)$$

$$\beta_3 = \sin^{-1} \frac{\{(\mathbf{b}_6 - \mathbf{b}_5) \times \mathbf{k}\} \times \mathbf{i}}{|(\mathbf{b}_6 - \mathbf{b}_5) \times \mathbf{k}|} \quad (2.25)$$

where β_1 is the angle between the normal to b_1b_2 and the x axis, β_2 is the angle between the normal to b_3b_4 and the x axis, β_3 is the angle between the normal to b_5b_6 and the x axis, and **i**, **j** and **k** are the unit vectors of the x, y, and z axes, respectively. In a compact notation,

$$\beta_i = \sin^{-1} \frac{\{(\mathbf{b}_{2i} - \mathbf{b}_{2i-1}) \times \mathbf{k}\}, \mathbf{i}}{|(\mathbf{b}_{2i} - \mathbf{b}_{2i-1}) \times \mathbf{k}|} \qquad (i=1, 2, 3) \quad (2.26)$$

If the unit vectors in the O_1P_1, O_2P_2 and O_3P_3 directions are called $\mathbf{w}_1$, $\mathbf{w}_2$, and $\mathbf{w}_3$, respectively, and the angles between the $b_1b_2P_1$, $b_3b_4P_2$ and $b_5b_6P_3$ planes and the xy plane are called ϕ_1, ϕ_2, and ϕ_3, respectively, $\mathbf{w}_i$ $(i=1, 2, 3)$ vectors may be expressed as

$$\mathbf{w}_1 = \cos\beta_1 \cos\phi_1 \mathbf{i} + \sin\beta_1 \cos\phi_1 \mathbf{j} + \sin\phi_1 \mathbf{k} \quad (2.27)$$

$$\mathbf{w}_2 = \cos\beta_2 \cos\phi_2 \mathbf{i} + \sin\beta_2 \cos\phi_2 \mathbf{j} + \sin\phi_2 \mathbf{k} \quad (2.28)$$

$$\mathbf{w}_3 = \cos\beta_3 \cos\phi_3 \mathbf{i} + \sin\beta_3 \cos\phi_3 \mathbf{j} + \sin\phi_3 \mathbf{k} \quad (2.29)$$

In a compact form,

$$\mathbf{w}_i = \cos\beta_i \cos\phi_i \mathbf{i} + \sin\beta_i \cos\phi_i \mathbf{j} + \sin\phi_i \mathbf{k} \qquad (i=1, 2, 3) \quad (2.30)$$

Thus, the position vectors of points P_1, P_2 and P_3 of the top platform become

$$\mathbf{P}_1 = \mathbf{Q}_1 + m_1 \mathbf{w}_1 \quad (2.31)$$

$$\mathbf{P}_2 = \mathbf{Q}_2 + m_2 \mathbf{w}_2 \quad (2.32)$$

$$\mathbf{P}_3 = \mathbf{Q}_3 + m_3 \mathbf{w}_3 \quad (2.33)$$

In a compact form,

$$\mathbf{P}_i = \mathbf{Q}_i + m_i \mathbf{w}_i \qquad (i=1, 2, 3) \quad (2.34)$$

Since the top platform is a rigid body, position vectors above must satisfy the following constraints where L_{p1}, L_{p2} and L_{p3} are the lengths of the sides of the moving platform.

$$|\mathbf{P}_1 - \mathbf{P}_2|^2 = L_{p3}^2 \quad \textbf{(2.35)}$$

$$|\mathbf{P}_2 - \mathbf{P}_3|^2 = L_{p1}^2 \quad \textbf{(2.36)}$$

$$|\mathbf{P}_1 - \mathbf{P}_3|^2 = L_{p2}^2 \quad \textbf{(2.37)}$$

Yurt (2002) states that there are two methods that can be followed from here on. One of the mis to turn the nonlinear set of equations with three unknowns given in (2.35)-(2.37) into a 16th order polynomial by variable transformations, and the other one is to solve the nonlinear set of equations by a numerical method. Out of the 16 possible solutions obtained by following the polynomial method, at most 12 are real solutions, and only one of the mise the real physical solution because only one solution satisfies the mechanical and geometrical constraints. It is tedious to solve the 16^{th} order polynomial and test the results to see which one of these satisfies the mechanical and geometrical constraints and thus is the real physical solution to the FK problem.

2.3 Inverse Kinematics Analysis

Inverse kinematics analysis is the problem of finding the leg lengths when the translation and rotation of the top platform with respect to the base platform are known. If the position of the top platform is given in terms of a rotation matrix $\mathbf{R}$ and a translation vector $\mathbf{t}$, vertices of the moving platform ($\mathbf{p}_i$) are expressed in terms of the coordinate system fixed to the base platform ($\mathbf{P}_i$) such as

$$\mathbf{P}_i = \mathbf{R}\,\mathbf{p}_i + \mathbf{t} \quad (i=1,2,..6) \quad \textbf{(2.38)}$$

Translation vector gives the distance between the centers of gravity of the coordinate systems on the base and the top platform. Leg length vectors $\mathbf{S}_i$'s may be found by using the expressions of the vertices of the top and the base platform in terms of the coordinate system fixed to the base

$$\mathbf{S}_i = \mathbf{P}_i - \mathbf{b}_i \quad (i=1,2,..6) \quad \textbf{(2.39)}$$

where $\mathbf{b}_i$'s denote the vertices of the base platform and $\mathbf{P}_i$'s denote the vertices of the top platform. Figure 2.8 gives the SPM model used in the IK analysis.

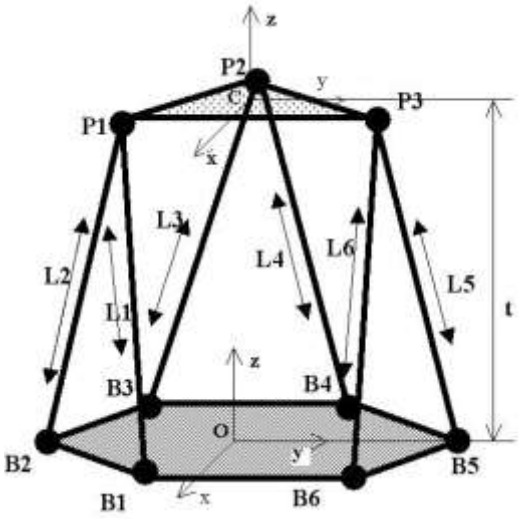


Figure 2.8: SPM model used in the IK analysis

The coordinates of the top platform with respect to the coordinate system fixed to the base platform are given by a rotation matrix, R, and a translation vector, t. In order to find R, the rotation matrix, the order of the rotations of the top platform with respect to the axes of the coordinate system fixed to its center of gravity need to be known. Assuming that the first rotation that the coordinate axes on the moving platform undergo is around the x axis, the second rotation is around the y axis, and the third is around the z axis, and that the magnitudes of the rotations are γ, β, and α degrees, respectively, the rotation matrix R is obtained through the use of Euler's angles as

$$R = (R_x \; R_y \; R_z)^T \qquad \textbf{(2.40)}$$

where R_x is the matrix of rotation about the x axis, R_y is the matrix of rotation about the y axis, and R_z is the matrix of rotation about the z axis. Taking into account that the rotation about x is γ degrees, the rotation about y is β degrees and rotation about z is α degrees, Euler's angles result in the following rotation matrices:

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & \sin\gamma \\ 0 & -\sin\gamma & \cos\gamma \end{bmatrix} \qquad \textbf{(2.41)}$$

$$R_y = \begin{bmatrix} \cos\beta & 0 & -\sin\beta \\ 0 & 1 & 0 \\ \sin\beta & 0 & \cos\beta \end{bmatrix} \quad (2.42)$$

$$R_z = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.43)$$

and finally

$$R = \begin{bmatrix} \cos\alpha\cos\beta & -\sin\alpha\cos\gamma + \cos\alpha\sin\beta\sin\gamma & \sin\alpha\sin\gamma + \cos\alpha\sin\beta\cos\gamma \\ \sin\alpha\cos\beta & \cos\gamma\cos\alpha + \sin\alpha\sin\beta\sin\gamma & -\sin\gamma\cos\alpha + \cos\gamma\sin\alpha\sin\beta \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma \end{bmatrix} \quad (2.44)$$

Derivation of the rotation matrices using Euler's angles is given in Section 4.3 in detail. In the 6-3 SPM, if P_i's are used to denote the position of the vertices of the top platform with respect to the coordinate system fixed to the base (Figure 2.8), and p_i's are used to denote the coordinates of the same vertices with respect to the coordinate system on the moving platform, the following relation can be written.

$$P_i = R\,p_i + t \quad (i = 1, 2, 3) \quad (2.45)$$

By using the rotation matrix R and translation vector t, the expressions of the coordinates of the vertices of the top platform are obtained in terms of the coordinate system fixed to the base platform. Translation vector t gives the offset between the centers of gravity of the coordinate systems fixed to the base and the top platform, in coordinates of the system fixed to the base. In other words, it is the vector describing the origin of the coordinate system on the top platform in coordinates of the system on the base platform. A quick inspection of Figure 2.9 reveals that leg vectors are found by taking the difference of the coordinates of the vertices of the base platform, namely B_j's, $(i = 1 \ldots 6)$ and the P_i's. Leg lengths, L_i's, are the norms of these vectors.

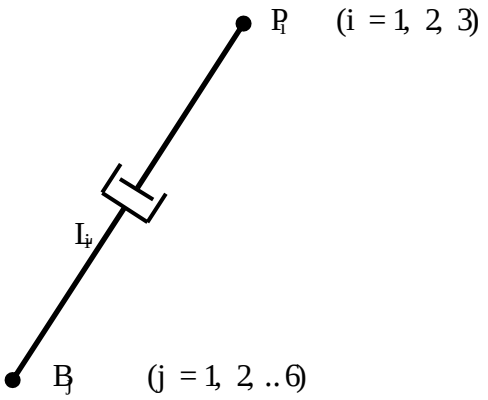


Figure 2.9: One leg of the SPM

It can easily be seen that the IK analysis of the 6-3 SPM is quite simple. Another important aspect of the IK analysis is that the IK analysis yields a unique result, meaning there is only one set of leg lengths that the SPM can have for a given position of the center of gravity of the moving platform and orientation of the moving platform with respect to the base platform.

3. NEURAL NETWORKS

Heuristic algorithms are self learning or adaptive processes that seek a solution among all possible solutions but do not guarantee that the optimal solution will be found. This makes heuristic algorithms approximate, rather than exact. Since such algorithms are self-learning, they get better with experience. The two most popular heuristic algorithms are neural networks and genetic algorithms. This thesis deals with neural networks, thus this chapter will explain neural networks in detail.

Neural networks originated from psychology, as a model of how the brain works. Neural networks are composed of simple elements operating in parallel. These elements, called neurons, are inspired by biological neural systems. As in nature, the network function is determined largely by the connections between elements. A neural network can be trained to perform a particular function by adjusting the values of the connection weights between elements. Since the mid 1960's, Stephen Grossberg has been developing a mathematical model of the brain's function. He has come up with several neural network models with the ability to do training online, capacity for self-organization and the ability to form compact representations of complex phenomena. Although McCulloch and Pitts formulated the first digital neuron in 1943, Donald Hebb was the first to state that changes in strengths of the synapses that connect neurons in a neural network affect the activations on the neurons. This formed the basis of a network with the ability to learn. Rosenblatt applied the theory of Hebbian learning to update synaptic weights in two-layer networks and formulated a learning rule where the weights are adjusted in proportion to the error between the output of a neuron and the target value. Rosenblatt's research was a giant step in the area of neural networks because he proved that updating synaptic weights in such a way results in a desired set of weights. Next step in the study of neural networks was multilayer networks since many problems could not be solved with two-layer networks [84]. Figure 3.1 gives the schematic of a two layer network.

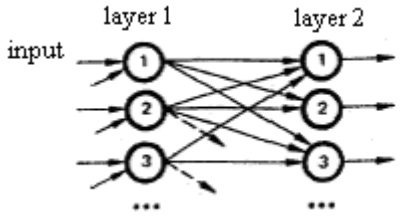


Figure 3.1: A two-layer network

Werbos and Parker discovered backpropagation independently in 1974 and 1982, respectively, which was the next major step in the advancement of neural networks, because backpropagation allows the training of multilayer networks. It is a powerful and practical tool for solving problems that would be quite difficult to solve otherwise. Neural networks (NN's) are adopted as a problem-solving tool in many branches of engineering [83]. Artificial NN's classify input patterns and adapt to pattern populations. They have numerous applications because they replace expensive programming efforts with simple self-training hardware. Applications of neural networks include, but are not limited to pattern recognition, decision making, prediction, and function generation, such as in the case of robot kinematics, which is the core of this thesis [84]. Their main advantage lies in the fact that they can learn new things without having to be explicitly programmed [86].

3.1 Advantages of Neural Networks

Advantages of neural network are listed below

1) Neural networks remove the need to formulate the mathematical model representing the relationship between the factors that are correlated. Neural networks take the data they are provided with and determine what part of the data is relevant. Irrelevant data has low connection strength so that it has no effect in the output neuron. Thus, relevant factors do not have to be determined beforehand.

2) For difficult problems, NN's are likely to be more accurate than any statistical model that may be formulated. There are many factors affecting the performance of a NN but NN's are sophisticated enough to solve complicated problems accurately.

3) A NN is a direct way of modeling a problem, compared to conventional mathematical modeling. Network trains itself by using the data patterns it is provided with, rather than solving the mathematical model. What is done is to represent both

input variables and output variables as neurons and literally connect them through at least one hidden layer of neurons.

4) Noise in the training data is not a problem in NN's since millions of data may be used in training.

5) Since the synapses and neurons in the same layer do not have any time dependencies, they may operate synchronously, which makes NN's suitable for parallel programming applications. NN applications, such as the NN code written for this thesis, run successfully on single processor computers, as well [83].

6) NN technology offers the promise that computers can dynamically adapt to new situations, whereas in conventional programming, all the actions and data structures defining the model have to be declared by the human programmer. In other words, in conventional programming, the programmer is responsible from every aspect of the application. NN's, on the other hand, require nothing more than the appropriate learning algorithm and a set of data patterns representing the behavior of the network. The network learns the relations between the data patterns by altering a set of weight values contained in its structure.

7) NN's can be applied in various fields of science and engineering, including aerospace, automotive, industrial, manufacturing, and robotics applications.

3.2 Biological Neural Networks

Biological neural networks are made up of simple elements called neurons. Artificial neural networks used for computation and control mimic the properties of biological neural networks. When biological NN's are compared to artificial NN's, it is seen that biological decisions are quite slow compared to electronic logic, but living systems can make enormously complicated decisions, involving millions of neurons, which is practically impossible in the case of designing artificial NN's. Modeling living neural networks involves a great amount of simplification.

Biological neurons, or nerve cells, exchange information with other neurons through chemical changes at synapses, which are microscopic gaps between neighboring neurons. Figure 3.2 shows a neuron transmitting information between other neurons. Input synapses of a nerve cell are at the ends of branches called dendrites and output synapses are found at ends of nerve fibers called axons. It must be noted here that a biological neuron may have thousands of connections. Inputs and outputs are

transmitted through the release of chemicals called transmitters. Transmitters react with the corresponding receptor chemicals on the receiving part of a synapse. The membrane permeability to various types of ions is changed, and thus the voltage near the axon is modified. Neuron electrochemistry adds or subtracts the inputs to an individual neuron, all with different weights. Consequently, a voltage propagates down the axon and produces a chemical output by releasing a chemical transmitter at the output synapse.

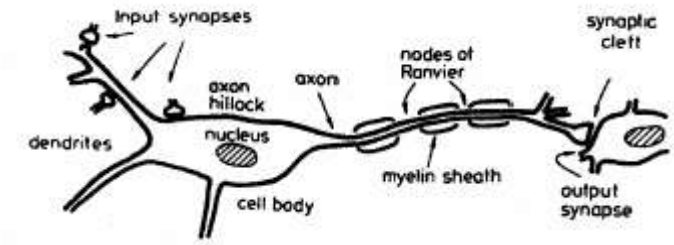


Figure 3.2: Anatomy of a neuron

Biological neurons turn on and off at a rate of about 1000 times per second, where as computers can switch over a billion times a second. Despite this vast difference in operation speed, humans can solve some difficult problems that computers cannot, due to the architectural differences the human brain and the computer. The human brain can process an immense amount of information in parallel, while a computer has to process either fully or partially sequentially [84].

3.3 Artificial Neural Networks

An artificial neural network is a collection of parallel processors, called neurons, connected through links, inspired by the nature. In Figure 3.3, the nodes represent neurons, arrows represent connections and the directions of the arrows represent the direction of signal flow. Each neuron performs a summation of its inputs to determine its activation. Neurons are grouped together in layers. Processing of information in a neural network begins with the application of a set of signals to the input layer, denoted by the $i_1, \ldots i_n^{th}$ neurons. $o_1, o_2, \ldots o_q$ represent the neurons in the output layer. Hidden layer weight matrix and output layer weight matrix are represented by w1 and w2, respectively. For both matrices, element (i,j) represents the connection weight value between the i^{th} neuron of the preceding layer and j^{th}

neuron of the latter layer. Each signal in the input set stimulates one of the neurons in the input layer and each of the neurons in the input layer produces one output signal. The outputs produced by the processing elements, namely the neurons in a layer are passed on to the next layer of neurons. This process is repeated until the last layer of neurons, denoted by $o_1, \ldots o_q$, produces an output for the input-pattern vector presented to the network [86].

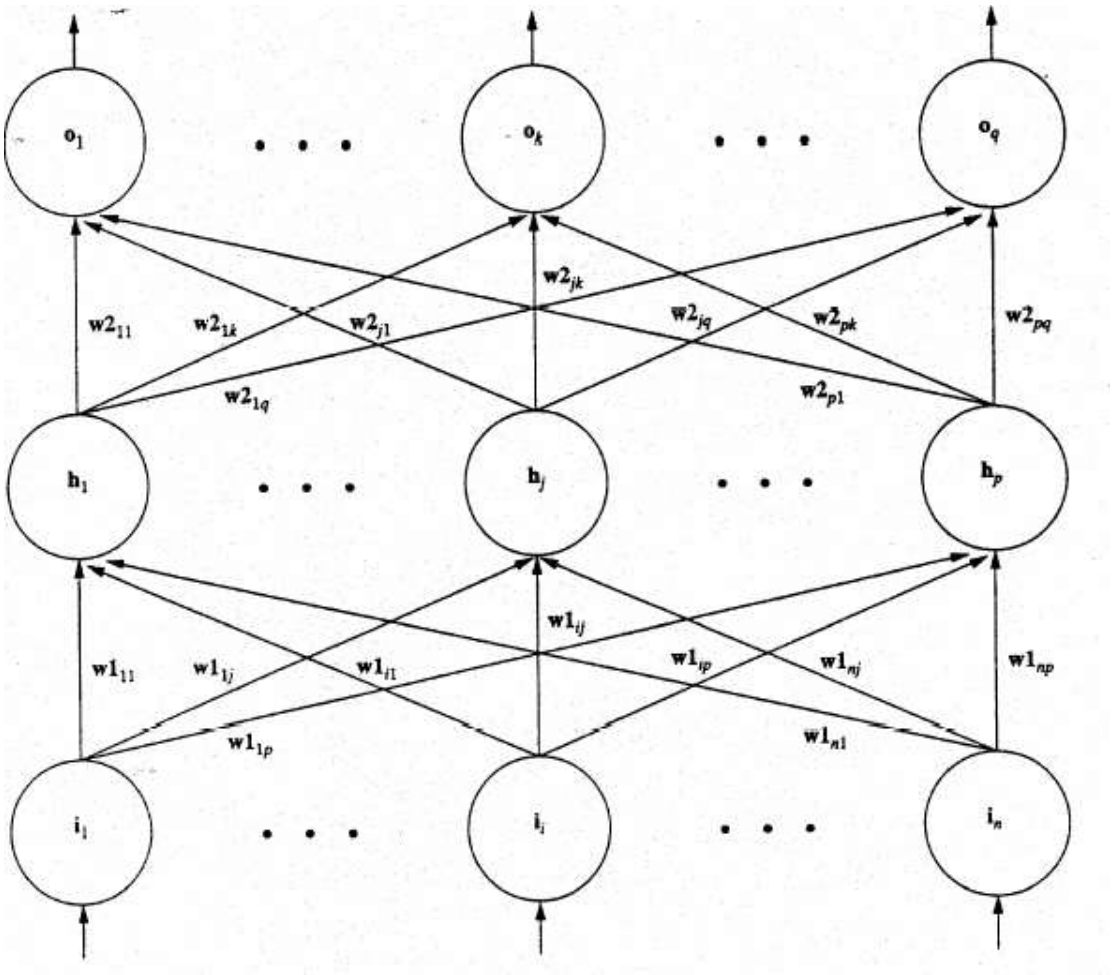


Figure 3.3: Diagram of a NN model

Artificial NN's may be composed of different numbers of neurons. The size of the network, meaning the number of layers and the number of neurons in each layer depends on the application. The NN application given in this thesis has six neurons in the input layer and six neurons in the output layer, due to the nature of the FK problem of the 6-3 SPM. Since the FK problem solves for six values, which are the position of the center of gravity of the moving platform of the SPM and the rotation of the moving platform with respect to the base platform when six leg lengths are

given, it means that the NN needs to produce six outputs in the presence of six inputs. Thus, the architecture has six neurons in both the input and the output layers.

NN's frequently have one or two layers of neurons between the input and output layers, though they may have more than two layers of neurons between the input and the output layers. The layers between the input and the output layers are called hidden layers because they do not interact directly with the outside, whereas the input layer receives signals from the outside and the output layer outputs signals visible to the outside world. There is no mathematical rule to determine the optimal number of hidden layers or the optimal number of neurons in the hidden layers for an application [85]. The number of neurons in the input and output layers are fixed in a problem, though. The number of hidden layers and the number of neurons in the hidden layers are usually determined by trial-and-error. The network function is determined by the connection weight values between elements called neurons and biases. A NN is trained to produce certain outputs for certain inputs. It is possible for a NN to realize a function by changing its weights and biases.

3.3.1 Structure of a neuron

A neuron is the fundamental building block, or the basic processor, of a neural network. Artificial neurons treat data similar to biological neurons exposed to incoming signals. Figure 3.4 compares a biological processing unit with an artificial processing unit.

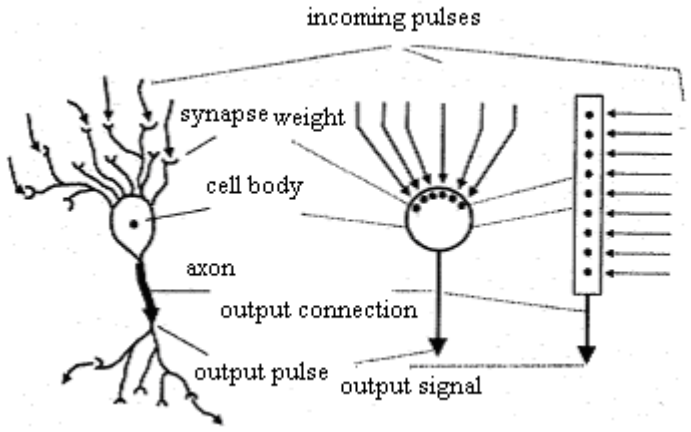


Figure 3.4: Comparison of a biological neuron and an artificial neuron

Each artificial neuron receives several input signals from neurons belonging to the preceding layer over connections, except for the input layer of neurons. These inputs

are the activations of the incoming neurons multiplied by the synaptic weights. These products are combined to form a combined input signal to the neuron. It should be noted here that magnitudes of both the incoming signals and the synaptic weights contribute to the input to a neuron. The output of a neuron is computed by applying a threshold function to this input signal plus a bias term so that the output value is confined within a finite interval. The network changes these synaptic weights and biases through the self-training process so that input signals can have relatively large or small effects on the processing unit. The output of a neuron is obtained by applying a threshold function to the sum of the activations on the neuron and possibly a bias term. The figures below give neuron models. It can be observed that each neuron is connected only to neurons in the layers immediately before or after its own layer. Figure 3.5 depicts that each incoming activation signal is multiplied by the relevant synaptic weight and the activation function is applied to obtain the outgoing activation of a neuron. Figure 3.6 shows that more than signal may enter a neuron and the outgoing activation may enter more than neurons. In reality, the outgoing activation of a neuron enters all the neurons in the following layer.

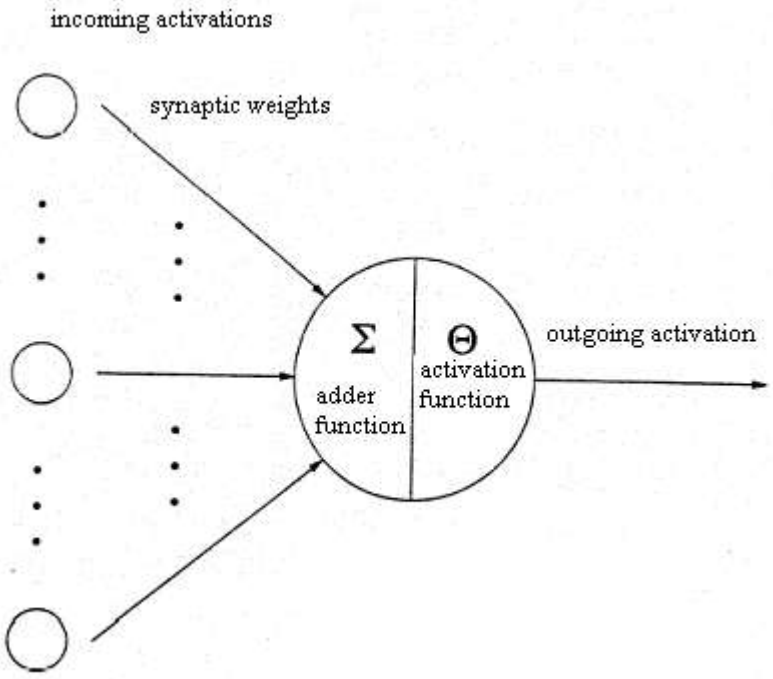


Figure 3.5: Diagram of a neuron

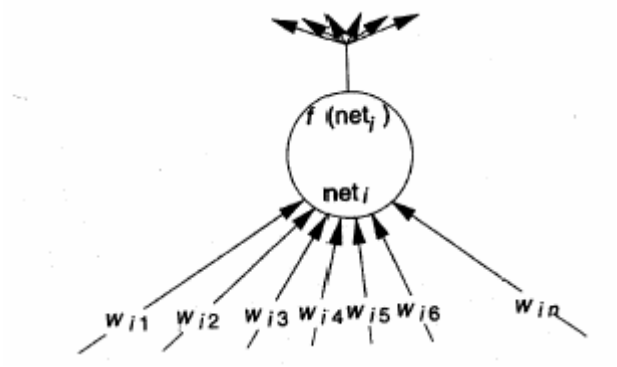


Figure 3.6: Signals coming in and out of a neuron

The computation performed by the neuron to determine its input stimulation is to multiply the signal through each connection by the connection weight value, sum those products and perhaps add a bias term. An activation function is applied to compute the output of the neuron [83, 85, 86]. Mathematical relations governing the activities of neurons and the entire NN are derived in sections 3.4.1 – 3.4.9.

3.3.2 Activation function

An activation function, or sometimes referred to as a transfer function, is used by a neuron to produce an output signal related to its activation. The activation function used by a network depends on the application. The activation function may be a linear function which simply sums every signal that comes into the unit, it may be a binary-threshold function with two stable states so that the network detects the presence or absence of certain features in the data pattern by becoming active or inactive (Figure 3.7), or it may be a sigmoidal function (Figure 3.8), which often is the case [86]. A binary threshold function is used to impart the active or inactive states of a unit. It can be given as

$$f(net(t)) = 1 \text{ if } net(t) > \theta \qquad \textbf{(3.1)}$$
$$0 \text{ otherwise}$$

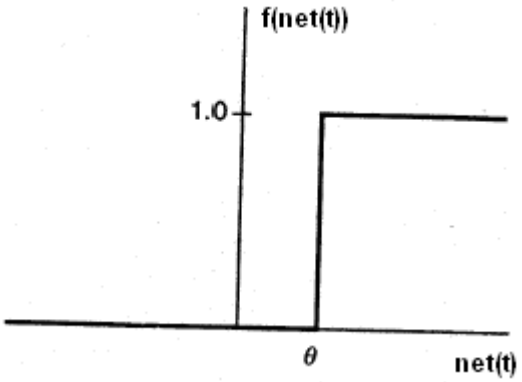


Figure 3.7: A binary threshold function

Once a transfer function is chosen for a particular network, it remains unchanged for all neurons in the network [85]. Examples of sigmoidal functions are the logistic function, given as

$$f(x) = \frac{1}{1+e^{-x}} \tag{3.2}$$

and shown graphically in Figure 3.8 or the tangent-sigmoid function, which is mathematically equivalent to the tangent-hyperbolic function, given as

$$f(x) = \frac{2}{1+e^{-2x}} - 1 \tag{3.3}$$

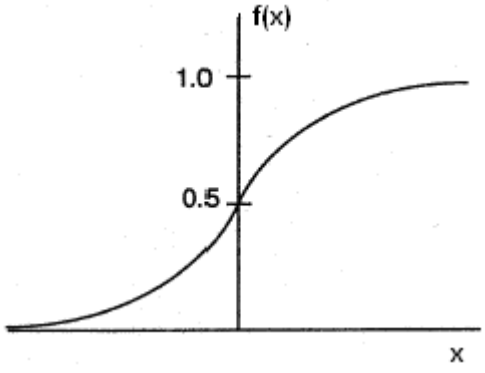


Figure 3.8: Logistic sigmoid function

A non-linear activation function should be used to produce non-linear mappings from the input space to the output space in a NN [83].

3.3.3 Learning

The knowledge that a NN possesses is stored in the synapses, in other words, the connection weights of the neurons and biases. A NN will produce the right output for an input if the set of synaptic weights and biases are right. The network gains this knowledge through the process called training. Data patterns are presented to the network sequentially, the difference between the output signal produced by the network and the target output is observed, and finally the weights and biases are updated by taking the error in the output pattern into consideration [83]. It is the network itself that finds the appropriate weights and biases to solve a particular problem, thus it is said to be learning to produce a desired transformation [86].

3.4 Backpropagation Algorithm

Backpropagation of errors is the most commonly used neural-network training method. Backpropagation involves the computation of errors in each layer of the neural network and propagation of errors backwards through the network in order to update weights and biases. This strategy is based on comparing the output of the model and the actual answer that should be produced, in simple terms, it is based on error minimization. A "good" set of data patterns must be present for training. It should be taken into consideration that for backpropagation to yield fine results, the topology of the NN, meaning the number and size of the hidden layers should be carefully chosen. Unfortunately, there is no mathematical rule to determine those values, thus they must be found by trial-and-error.

A backpropagation network learns to produce a mapping from the input pattern to the output pattern by minimizing the error between the output of the NN and the target value. A data pattern is introduced to the network, and it is propagated through the network to produce an output pattern. This makes up the feedforward part of the learning process. In the backpropagation part, the error between the output produced by the network and the target pattern is propagated backwards in the network and is used to alter the connection weight values and biases. These steps are repeated for all data patterns, until the error between the output and the target values falls below an acceptable limit. Sections 3.4.1 – 3.4.4 constitute the feedforward part of the

algorithm, sections 3.4.5 – 3.4.7 constitute the backpropagation of errors and sections 3.4.8.1 – 3.4.8.6 constitute the training part of the algorithm.

Consider a NN architecture with n_input neurons in the input layer, n_first_hidden neurons in the first hidden layer, n_second_hidden neurons in the second hidden layer and n_output neurons in the output layer, as given in Figure 3.9.

In this case, the dimensions of the weights and biases are given in Table 3.1 as the following.

Table 3.1: Dimensions of the NN

Variable Name	Explanation	Size
weight1	first hidden layer weight matrix	(n_first_hidden, n_input)
weight2	second hidden layer weight matrix	(n_second_hidden, n_first_hidden)
weight3	output layer weight matrix	(n_output, n_second_hidden)
bias1	first hidden layer bias	(1, n_first_hidden)
bias2	second hidden layer bias	(1, n_second_hidden)
bias3	output layer bias	(1, n_output)

NN training has three parts, the feedforward part, the backpropagation part and the updating of weights and biases. In the feedforward part, signals entering the input neurons propagate through the first hidden layer, second hidden layer and finally the output layer neurons to produce certain outputs. In the backpropagation part, output layer error, second hidden layer error, first hidden layer error and total error are calculated, respectively, and finally in the updating part the weight matrices and biases are updated so as to decrease the total error between the output produced by the network and the actual target value that is supposed to be produced. NN training terminates when the total error is minimized or reaches an acceptable level. The following are the steps to follow during NN training. Before training starts, random values are assigned to the weight and bias matrices. Sections 3.4.1 – 3.4.4 make up the feedforward part of the algorithm presented here.

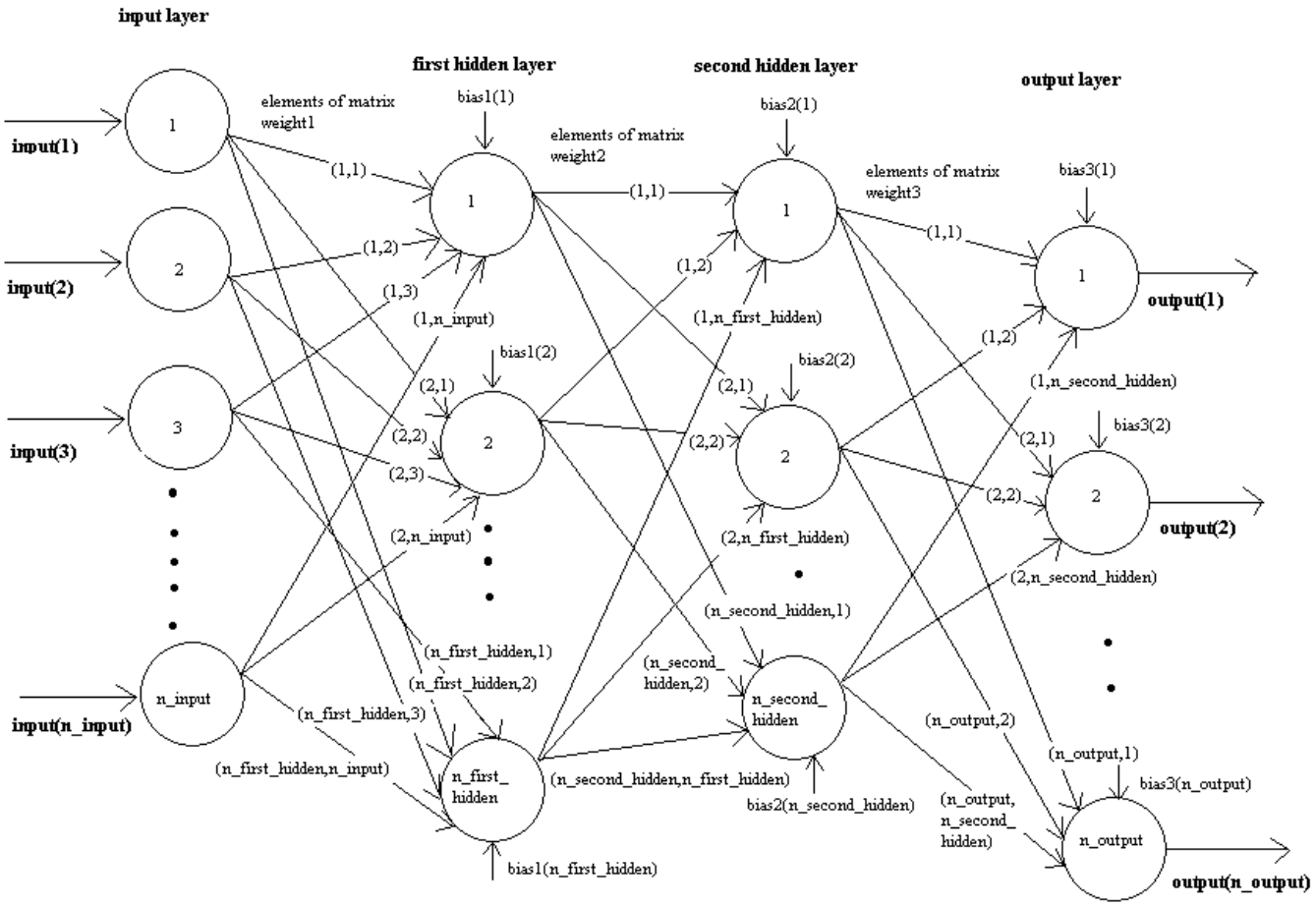


Figure 3.9: Diagram of a NN with two hidden layers

3.4.1 Selection of one data pattern

One training data pair is selected from the NN training data set. Training data can be saved in the form of row matrices, thus, the input patterns have size (1, n_input) while the target patterns have size (1, n_output) where n_input is the number of neurons in the input layer of the NN and n_output is the number of neurons in the output layer of the NN. Numbers of neurons in the two hidden layers do not depend on the number of neurons in the input or output layers. Their optimal value can only be found on the basis of trial-and-error. Having too many neurons in the hidden layer does not promise better convergence. Weights and biases may be generated randomly before training starts since they will be constantly updated during the backpropagation process.

3.4.2 Activation on the first hidden layer

The selected data pattern, first hidden layer weight matrix and first hidden later bias are used to compute the activation on the first hidden layer. The activation on the first hidden layer can be represented as a matrix of size (1, n_first_hidden) where n_first_hidden is the number of neurons in the first hidden layer. Table 3.2 gives the values used in calculating the activation on the first hidden layer.

Table 3.2: Calculation of the activation on the first hidden layer

Name	Description	Size
n_input	number of neurons in the input layer	1
n_first_hidden	number of neurons in the first hidden layer	1
input	input data pattern	(1, n_input)
weight1	first hidden layer weight matrix	(n_first_hidden, n_input)
bias1	first hidden layer bias	(1, n_first_hidden)
hidden1	activation on first hidden layer	(1, n_first_hidden)
hidden1_log	output of the first hidden layer	(1, n_first_hidden)

After a quick inspection of the NN diagram in Figure 3.9, the activation on each neuron in the first hidden layer can be written as the following.

hidden1(1,1) = input(1,1) * weight1(1,1) + input(1,2) * weight1(1,2)

$$+ input(1,3) * weight1(1,3) + \ldots$$
$$+ \ldots input(1,n\_input) * weight1(1,n\_input) + bias1(1,1) \quad (3.4)$$

$$hidden1(1,2) = input(1,1) * weight1(2,1) + input(1,2) * weight1(2,2) +$$
$$+ input(1,3) * weight1(2,3) + \ldots$$
$$+ \ldots input(1,n\_input) * weight1(2,n\_input) + bias1(1,2) \quad (3.5)$$

$$hidden1(1,3) = input(1,1) * weight1(3,1) + input(1,2) * weight1(3,2)$$
$$+ input(1,3) * weight1(3,3) + \ldots$$
$$+ \ldots input(1,n\_input) * weight1(3,n\_input) + bias1(1,3) \quad (3.6)$$

.

.

$$hidden1(1,n\_first\_hidden) = input(1,1) * weight1(n\_first\_hidden,1)$$
$$+ input(1,2) * weight1(n\_first\_hidden,2)$$
$$+ input(1,3) * weight1(n\_first\_hidden,3) + \ldots$$
$$+ \ldots input(1,n\_input) * weight1(n\_first\_hidden,n\_input)$$
$$+ bias1(1,n\_first\_hidden) \quad (3.7)$$

In matrix notation,

$$[hidden1] = [input] * [weight1]^T + [bias1] \quad (3.8)$$

These are the activations arriving at each neuron in the first hidden layer. The outputs of these neurons are obtained when the activation function is applied to these values. In this application, the activation function is chosen to be the tangent-sigmoid function, which is denoted as

$$F(x) = \frac{2}{1 + e^{-2x}} - 1 \quad (3.9)$$

Thus, the output of each neuron in the first hidden layer can be shown as

$$hidden1\_log(1,1) = \frac{2}{1 + e^{-2(hidden1(1,1))}} - 1 \quad (3.10)$$

$$hidden1\_log(1,2) = \frac{2}{1 + e^{-2(hidden1(1,2))}} - 1 \quad (3.11)$$

$$hidden1\_log(1,3) = \frac{2}{1 + e^{-2(hidden1(1,3))}} - 1 \quad \textbf{(3.12)}$$

.

.

$$hidden1\_log(1,n\_first\_hidden) = \frac{2}{1 + e^{-2(hidden1(1,n\_first\_hidden))}} - 1 \quad \textbf{(3.13)}$$

In matrix notation,

$$[hidden1\_log] = F([hidden1]) = F([input] * [weight1]^T + [bias1]) \quad \textbf{(3.14)}$$

3.4.3 Activation on the second hidden layer

The outputs of the first hidden layer neurons, second hidden layer weight matrix and second hidden layer bias are used to compute the activation on the second hidden layer. Outputs of the first hidden layer neurons are the inputs to the second hidden layer neurons. The activation on the second hidden layer can be represented as a matrix of size (1, n_second_hidden) where n_second_hidden is the number of neurons in the second hidden layer. Table 3.3 gives the values used in calculating the activation on the second hidden layer.

Table 3.3: Calculation of the activation on the second hidden layer

Name	Description	Size
n_first_hidden	number of neurons in the first hidden layer	1
n_second_hidden	number of neurons in the second hidden layer	1
hidden1_log	input to the second hidden layer, output of the first hidden layer	(1, n_first_hidden)
weight2	second hidden layer weight matrix	(n_second_hidden, n_first_hidden)
bias2	second hidden layer bias	(1, n_second_hidden)
hidden2	activation on second hidden layer	(1, n_second_hidden)
hidden2_log	output of the second hidden layer	(1, n_second_hidden)

After a quick inspection of the NN diagram in Figure 3.9, the activation on each neuron in the second hidden layer can be written as the following,

$$\begin{aligned}hidden2(1,1) &= hidden1\_log(1,1) * weight2(1,1) \\ &+ hidden1\_log(1,2) * weight2(1,2) \\ &+ hidden1\_log(1,3) * weight2(1,3) + \ldots \\ &+ \ldots hidden1\_log(1,n\_first\_hidden) * weight2(1,n\_first\_hidden) \\ &+ bias2(1,1)\end{aligned} \quad \textbf{(3.15)}$$

$$\begin{aligned}hidden2(1,2) &= hidden1\_log(1,1) * weight2(2,1) \\ &+ hidden1\_log(1,2) * weight2(2,2) + \\ &+ hidden1\_log(1,3) * weight2(2,3) + \ldots \\ &+ \ldots hidden1\_log(1,n\_first\_hidden) * weight2(2,n\_first\_hidden) \\ &+ bias2(1,2)\end{aligned} \quad \textbf{(3.16)}$$

$$\begin{aligned}hidden2(1,3) &= hidden1\_log(1,1) * weight2(3,1) \\ &+ hidden1\_log(1,2) * weight2(3,2) \\ &+ hidden1\_log(1,3) * weight2(3,3) + \ldots \\ &+ \ldots hidden1\_log(1,n\_first\_hidden) * weight2(3,n\_first\_hidden) \\ &+ bias2(1,3)\end{aligned} \quad \textbf{(3.17)}$$

.

.

$$\begin{aligned}hidden2(1,n\_second\_hidden) &= hidden1\_log(1,1) * weight2(n\_second\_hidden,1) \\ &+ hidden1\_log(1,2) * weight2(n\_second\_hidden,2) \\ &+ hidden1\_log(1,3) * weight2(n\_second\_hidden,3) + \ldots \\ &+ hidden1\_log(1,n\_first\_hidden) * \\ &\quad weight2(n\_second\_hidden,n\_first\_hidden) \\ &+ bias2(1,n\_second\_hidden)\end{aligned} \quad \textbf{(3.18)}$$

In matrix notation,

$$[hidden2] = [hidden1\_log] * [weight2]^T + [bias2] \quad \textbf{(3.19)}$$

These are the activations arriving at each neuron in the second hidden layer. The outputs of these neurons are obtained when the activation function is applied to these values. Thus, the output of each neuron in the second hidden layer can be shown as

$$hidden2\_log(1,1) = \frac{2}{1 + e^{-2(hidden2(1,1))}} - 1 \quad \textbf{(3.20)}$$

$$hidden2\_log(1,2) = \frac{2}{1 + e^{-2(hidden2(1,2))}} - 1 \quad \textbf{(3.21)}$$

$$hidden2\_log(1,3) = \frac{2}{1 + e^{-2(hidden2(1,3))}} - 1 \quad \textbf{(3.22)}$$

.

.

$$hidden2\_log(1,n\_second\_hidden) = \frac{2}{1 + e^{-2(hidden2(1,n\_second\_hidden))}} - 1 \quad \textbf{(3.23)}$$

In matrix notation,

$$[hidden2\_log] = F([hidden2]) = F([hidden1\_log] * [weight2]^T + [bias2]) \quad \textbf{(3.24)}$$

3.4.4 Activation on the output layer

The outputs of the second hidden layer neurons, output layer weight matrix and output layer bias are used to compute the activation on the output layer. Outputs of the second hidden layer neurons are the inputs to the output layer neurons. The activation on the output layer can be represented as a matrix of size (1, n_output) where n_output is the number of neurons in the output layer. Table 3.4 gives the values used in calculating the activation on the output layer.

After a quick inspection of the NN diagram in Figure 3.9, the activation on each neuron in the output layer can be written as the following.

$$\begin{aligned} output(1,1) = {} & hidden2\_log(1,1) * weight3(1,1) + hidden2\_log(1,2) * weight3(1,2) \\ & + hidden2\_log(1,3) * weight3(1,3) + \ldots \\ & + \ldots hidden2\_log(1,n\_second\_hidden) * \\ & weight3(1,n\_second\_hidden) \\ & + bias3(1,1) \end{aligned} \quad \textbf{(3.25)}$$

Table 3.4: Calculation of the activation on the output layer

Name	Description	Size
n_second_hidden	number of neurons in the second hidden layer	1
n_output	number of neurons in the output layer	1
hidden2_log	input to the output layer, output of the second hidden layer	(1, n_second_hidden)
weight3	output layer weight matrix	(n_output, n_second_hidden)
bias3	output layer bias	(1, n_output)
output	activation on output layer	(1, n_output)
output_log	output of the neural network	(1, n_output)

$$\begin{aligned} output(1,2) = {} & hidden2\_log(1,1) * weight3(2,1) + hidden2\_log(1,2) * weight3(2,2) \\ & + hidden2\_log(1,3) * weight3(2,3) + \ldots \\ & + \ldots hidden2\_log(1,n\_second\_hidden) * \\ & weight3(2,n\_second\_hidden) \\ & + bias3(1,2) \end{aligned} \quad \textbf{(3.26)}$$

$$\begin{aligned} output(1,3) = {} & hidden2\_log(1,1) * weight3(3,1) + hidden2\_log(1,2) * weight3(3,2) \\ & + hidden2\_log(1,3) * weight3(3,3) + \ldots \\ & + \ldots hidden2\_log(1,n\_second\_hidden) * \\ & weight3(3,n\_second\_hidden) \\ & + bias3(1,3) \end{aligned} \quad \textbf{(3.27)}$$

.

.

$$\begin{aligned} output(1,n\_output) = {} & hidden2\_log(1,1) * weight3(n\_output,1) \\ & + hidden2\_log(1,2) * weight3(n\_output,2) \\ & + hidden2\_log(1,3) * weight3(n\_output,3) + \ldots \\ & + \ldots hidden2\_log(1,n\_second\_hidden) * \\ & weight3(n\_output,n\_second\_hidden) \\ & + bias3(1,n\_output) \end{aligned} \quad \textbf{(3.28)}$$

In matrix notation,

$$[output] = [hidden2\_log] * [weight3]^T + [bias3] \quad \textbf{(3.29)}$$

These are the activations arriving at each neuron in the output layer. The outputs of these neurons are obtained when the activation function is applied to these values. Thus, the output of each neuron in the output layer can be shown as

$$output\_log(1,1) = \frac{2}{1 + e^{-2(output(1,1))}} - 1 \quad \textbf{(3.30)}$$

$$output\_log(1,2) = \frac{2}{1 + e^{-2(output(1,2))}} - 1 \quad \textbf{(3.31)}$$

$$output\_log(1,3) = \frac{2}{1 + e^{-2(output(1,3))}} - 1 \quad \textbf{(3.32)}$$

.

.

$$output\_log(1,n\_output) = \frac{2}{1 + e^{-2(output(1,n\_output))}} - 1$$

In matrix notation,

$$[output\_log] = F([output]) = F([hidden2\_log] * [weight3]^T + [bias3]) \quad \textbf{(3.33)}$$

The steps covered in sections 3.4.1 – 3.4.4 constitute the feedforward part of the NN. From this point on, backpropagation starts. Sections 3.4.5 – 3.4.7 describe the backpropagation of errors.

3.4.5 Computation of the output layer error

Output layer errors are calculated using the output produced by the NN, the actual target value and the derivative of the activation function in the output layer. It must be noted that since the output of the tangent-sigmoid function has to be in the interval [-1, 1], the target values used in NN training must be scaled to lie within that range, as well. Once the feedforward part is complete and the output layer errors are being calculated, however, the errors produced by the NN must be scaled to lie in their original range. Each element of the target matrix represents a different physical quantity, thus each one has a different range. It is best to scale each column of the target matrix separately before training starts so that the minimum value of each

column is -1 and the maximum value of each column is 1. When d, the output layer error, is being calculated, the difference (target(i)-output(i)) (i=1, 2,.., n_output) must go through a reverse scaling so that elements of the difference lie within the actual ranges instead of within [-1, 1]. Scaling has to be done because the output of the NN is in the range [-1, 1], due to the nature of the tangent-sigmoid function. Once the output error is calculated in the range [-1, 1], it must be scaled back to its actual range. Matrix called result is obtained by scaling matrix output_log back to the original range. Table 3.5 gives the values used in calculating the output layer error.

Table 3.5: Calculation of output layer error

Name	Description	Size
n_output	number of neurons in the output layer	1
d	output layer error	(1, n_output)
target	target data pattern	(1, n_output)
result	output produced by the nn	(1, n_output)
F^1(output)	derivative of the activation function in the output layer	(1, n_output)

The relation that gives the output layer error is

$$d(1,i) = (target(1,i)-result(1,i))\ F^1(output(1,i)),\ (i = 1, 2,.. n\_output) \quad \textbf{(3.34)}$$

where d(1,i) is the error at the i^{th} output neuron,

target(1,i) is the actual value that the i^{th} output neuron must produce,

result(1,i) is the output produced by the i^{th} output neuron and

F^1 is the derivative of the activation function in the output layer.

Target value is already known, the output matrix is calculated in section 3.4.4, result matrix is obtained by scaling output_log back to its original range and the derivative of the activation function is

$$F^1(x) = 1 - (F(x))^2 \quad \textbf{(3.35)}$$

$$F^1(output(i)) = 1 - (F(output(i)))^2 \quad \textbf{(3.36)}$$

Since

$$F(output(i)) = output\_log(i) \quad (3.37)$$

the output layer error becomes

$$d(i) = (target(i)-result(i)) * (1-(output\_log(i))^2),\ i = 1, 2, .. n\_output \quad (3.38)$$

If the relation above is expanded into indicial notation,

$$d(1) = (target(1)-result(1)) * (1-(output\_log(1))^2) \quad (3.39)$$

$$d(2) = (target(2)-result(2)) * (1-(output\_log(2))^2) \quad (3.40)$$

$$d(3) = (target(3)-result(3)) * (1-(output\_log(3))^2) \quad (3.41)$$

.

.

$$d(n\_output) = (target(n\_output)-result(n\_output)) * (1-(output\_log(n\_output))^2) \quad (3.42)$$

3.4.6 Computation of the second hidden layer error

Second hidden layer errors are calculated using the output layer error, the weight matrix of the output layer and the derivative of the activation function in the second hidden layer. It can be observed here that in the backpropagation algorithm, errors are transmitted backwards in each layer. It other words, second hidden layer error depends on the output layer error and the connection weight values between the second hidden set of neurons and output layer of neurons. Additionally, the derivative of the activation function operates on the output values of the second hidden layer of neurons. Therefore, it can be stated that the information that is needed in order to calculate the errors in the second hidden layer lies between the second hidden layer of neurons and the output layer of neurons. Table 3.6 gives the values used in calculating the second hidden layer error.

The mathematical relation that gives the second hidden layer error is

$$e(1,i) = F^1(hidden2(1,i)) \sum_{k=1}^{n\_output} d(1,k)\ weight3(k,i) \quad (3.43)$$

$$(i = 1, 2, .. n\_second\_hidden)$$

$$(k=1, 2, .. n\_output)$$

Table 3.6: Calculation of second hidden layer error

Name	Description	Size
n_second_hidden	number of neurons in the second hidden layer	1
n_output	number of neurons in the output layer	1
e	second hidden layer error	(1, n_second_hidden)
d	output layer error	(1, n_output)
weight3	output layer weight matrix	(n_output, n_second_hidden)
F^1(hidden2)	derivative of the activation function in the second hidden layer	(1, n_second_hidden)

where e(1,i) is the error at the i^{th} neuron of the second hidden layer,

F^1 is the derivative of the activation function in the second hidden layer,

d(1,k) is the error at the k^{th} output neuron and

weight3 is the output layer weight matrix. Output layer error is calculated in section 3.4.5, weight3 is already known and

$$F^1(\text{hidden2}(1,i)) = 1 - (F(\text{hidden2}(1,i)))^2 \quad \textbf{(3.44)}$$

Since

$$F(\text{hidden2}(1,i)) = \text{hidden2\_log}(1,i) \quad \textbf{(3.45)}$$

the second hidden layer error becomes

$$e(1,i) = (1-(\text{hidden2\_log}(1,i))^2) \sum_{k=1}^{n\_output} d(1,k)\ \text{weight3}(k,i) \quad \textbf{(3.46)}$$

$$(i = 1, 2, .. n\_second\_hidden)$$

$$(k=1, 2, .. n\_output)$$

If the relation above is expanded into indicial notation,

$$e(1,1) = (1-(\text{hidden2\_log}(1,1))^2) \sum_{k=1}^{n\_output} d(1,k)\ \text{weight3}(k,1) \quad \textbf{(3.47)}$$

$$= (1-(\text{hidden2\_log}(1,1))^2)$$

$$(d(1,1) * \text{weight3}(1,1) + d(1,2) * \text{weight3}(2,1)$$

$$+ d(1,3) * \text{weight3}(3,1) + \ldots + d(1,n\_output) * \text{weight3}(n\_output,1)) \quad \textbf{(3.48)}$$

$$e(1,2) = (1-(hidden2\_log(1,2))^2) \sum_{k=1}^{n\_output} d(1,k)\ weight3(k,2) \quad \textbf{(3.49)}$$

$$= (1-(hidden2\_log(1,2))^2)$$
$$(d(1,1) * weight3(1,2) + d(1,2) * weight3(2,2)$$
$$+ d(1,3) * weight3(3,2) + \ldots + d(1,n\_output) * weight3(n\_output,2)) \quad \textbf{(3.50)}$$

$$e(1,3) = (1-(hidden2\_log(1,3))^2) \sum_{k=1}^{n\_output} d(1,k)\ weight3(k,3) \quad \textbf{(3.51)}$$

$$= (1-(hidden2\_log(1,3))^2)$$
$$(d(1,1) * weight3(1,3) + d(1,2) * weight3(2,3)$$
$$+ d(1,3) * weight3(3,3) + \ldots + d(1,n\_output) * weight3(n\_output,3)) \quad \textbf{(3.52)}$$

.

.

$$e(1,n\_second\_hidden) = (1-(hidden2\_log(1,n\_second\_hidden))^2)$$
$$\sum_{k=1}^{n\_output} d(1,k)\ weight3(k,n\_second\_hidden) \quad \textbf{(3.53)}$$

$$= (1-(hidden2\_log(1,\ n\_second\_hidden))^2)$$
$$(d(1,1) * weight3(1,\ n\_second\_hidden)$$
$$+d(1,2) * weight3(2,\ n\_second\_hidden)$$
$$+d(1,3) * weight3(3,n\_second\_hidden) + \ldots$$
$$+d(1,n\_output) * weight3(n\_output,\ n\_second\_hidden)) \quad \textbf{(3.54)}$$

3.4.7 Computation of the first hidden layer error

First hidden layer errors are calculated using the second hidden layer error, the weight matrix of the second hidden layer and the derivative of the activation function in the first hidden layer. It can be observed once more that in the backpropagation algorithm, errors are transmitted backwards in each layer. It other words, first hidden layer error depends on the second hidden layer error and the connection weight values between the first hidden set of neurons and second hidden set of neurons. Additionally, the derivative of the activation function operates on the output values of the first hidden layer of neurons. Therefore, it can be stated that the information that is needed in order to calculate the errors in the first hidden layer lies between the first hidden layer of neurons and the second hidden layer of neurons. Table 3.7 gives the values used in calculating the first hidden layer error.

Table 3.7: Calculation of first hidden layer error

Name	Description	Size
n_first_hidden	number of neurons in the first hidden layer	1
n_second_hidden	number of neurons in the second hidden layer	1
f	first hidden layer error	(1, n_first_hidden)
e	second hidden layer error	(1, n_second_hidden)
weight2	second hidden layer weight matrix	(n_second_hidden, n_first_hidden)
F^1(hidden1)	derivative of the activation function in the first hidden layer	(1, n_first_hidden)

The mathematical relation that gives the first hidden layer error is

$$f(1,i) = F^1(hidden1(1,i)) \sum_{k=1}^{n\_second\_hidden} e(1,k)\ weight2(k,i) \quad \textbf{(3.55)}$$

$(i = 1, 2, .. n\_first\_hidden)$

$(k=1, 2, .. n\_second\_hidden)$

where $f(1,i)$ is the error at the i^{th} neuron of the first hidden layer,

F^1 is the derivative of the activation function in the first hidden layer,

$e(1,k)$ is the error at the k^{th} neuron in the second hidden layer and

weight2 is the second hidden layer weight matrix. Second hidden layer error is calculated in section 3.4.6, weight2 is already known and

$$F^1(hidden1(1,i)) = 1 - (F(hidden1(1,i)))^2 \quad \textbf{(3.56)}$$

Since

$$F(hidden1(1,i)) = hidden1\_log(1,i) \quad \textbf{(3.57)}$$

the first hidden layer error becomes

$$f(1,i) = (1-(hidden1\_log(1,i))^2) \sum_{k=1}^{n\_second\_hidden} e(1,k)\ weight2(k,i) \quad \textbf{(3.58)}$$

$(i = 1, 2, .. n\_first\_hidden)$

$(k=1, 2, .. n\_second\_hidden)$

If the relation above is expanded into indicial notation,

$$f(1,1) = (1-(\text{hidden1\_log}(1,1))^2)\ \Sigma_{k=1}^{n\_second\_hidden}\ e(1,k)\ \text{weight2}(k,1) \quad \textbf{(3.59)}$$

$$= (1-(\text{hidden1\_log}(1,1))^2)$$
$$(e(1,1) * \text{weight2}(1,1) + e(1,2) * \text{weight2}(2,1)$$
$$+ e(1,3) * \text{weight2}(3,1) + \ldots$$
$$+ e(1,n\_second\_hidden) * \text{weight2}(n\_second\_hidden,1)) \quad \textbf{(3.60)}$$

$$f(1,2) = (1-(\text{hidden1\_log}(1,2))^2)\ \Sigma_{k=1}^{n\_second\_hidden}\ e(1,k)\ \text{weight2}(k,2) \quad \textbf{(3.61)}$$

$$= (1-(\text{hidden1\_log}(1,2))^2)$$
$$(e(1,1) * \text{weight2}(1,2) + e(1,2) * \text{weight2}(2,2)$$
$$+ e(1,3) * \text{weight2}(3,2) + \ldots$$
$$+ e(1,n\_second\_hidden) * \text{weight2}(n\_second\_hidden,2)) \quad \textbf{(3.62)}$$

$$f(1,3) = (1-(\text{hidden1\_log}(1,3))^2)\ \Sigma_{k=1}^{n\_second\_hidden}\ e(1,k)\ \text{weight2}(k,3) \quad \textbf{(3.63)}$$

$$= (1-(\text{hidden1\_log}(1,3))^2)$$
$$(e(1,1) * \text{weight2}(1,3) + e(1,2) * \text{weight2}(2,3)$$
$$+ e(1,3) * \text{weight2}(3,3) + \ldots$$
$$+ e(1,n\_second\_hidden) * \text{weight2}(n\_second\_hidden,3)) \quad \textbf{(3.64)}$$

.

.

$$f(1,n\_first\_hidden) = (1-(\text{hidden1\_log}(1,n\_first\_hidden))^2)$$
$$\Sigma_{k=1}^{n\_second\_hidden} e(1,k)\ \text{weight2}(k,n\_first\_hidden) \quad \textbf{(3.65)}$$

$$= (1-(\text{hidden1\_log}(1,n\_first\_hidden))^2)$$
$$(e(1,1) * \text{weight2}(1,n\_first\_hidden)$$
$$+ e(1,2) * \text{weight2}(2,n\_first\_hidden)$$
$$+ e(1,3) * \text{weight2}(3,n\_first\_hidden) + \ldots$$
$$+ e(1,n\_second\_hidden) * \text{weight2}(n\_second\_hidden,n\_first\_hidden)) \quad \textbf{(3.66)}$$

The steps covered in sections 3.4.5 – 3.4.7 constitute the backpropagation part of the NN. From this point on, updating of weights and biases starts. Sections 3.4.8.1 – 3.4.8.6 describe the training process.

3.4.8 Updating weights and biases

Connection weights and biases are updated so that the NN learns to recognize patterns in the data. Updating weights incorporates the error in that layer, input to that layer, the previous change in that weight matrix, and two factors called the learning rate and momentum factor. Updating biases incorporates the error in that layer, the previous change in the bias, learning rate and momentum factor. Momentum factor determines how much the previous amount of change in the weights and biases affects the current amount of change. Both the learning rate and the momentum factor typically take values within the range (0, 1). The optimal values of the learning rate and that of the momentum factor depend on the specific application, similar to the optimal values of the numbers of neurons in the hidden layers. Updating weights and biases is the training part of the NN. Sections 3.4.8.1 – 3.4.8.6 describe how weights and biases are updated.

3.4.8.1 Updating first hidden layer weight matrix

Change in the first hidden layer weight matrix depends on the first hidden layer error, the input to the first hidden layer, the previous amount of change in the first hidden layer weight matrix, the learning rate and the momentum factor. Table 3.8 gives the values used while updating the first hidden layer weight matrix.

Table 3.8: Updating the first hidden layer weight matrix

Name	Description	Size
n_input	number of neurons in the input layer	1
n_first_hidden	number of neurons in the first hidden layer	1
weight1	first hidden layer weight matrix	(n_first_hidden, n_input)
f	first hidden layer error	(1, n_first_hidden)
input	input data pattern	(1, n_input)
$\Delta weight1_{previous}$	previous amount of change in the first hidden layer weight matrix	(n_first_hidden, n_input)
η	learning rate	1
θ	momentum factor	1

Mathematically, weight1(i,j) is updated as

$$\text{weight1}(i,j) = \text{weight1}(i,j) + \eta f(i)\,\text{input}(j) + \theta\,\Delta\text{weight1}_{\text{previous}}(i,j) \quad \textbf{(3.67)}$$

$$(i=1,2,\ldots n\_first\_hidden)$$
$$(j=1,2,\ldots n\_input)$$

where η is the learning rate,

f is the first hidden layer error,

input is the input data pattern matrix,

θ is the momentum factor and

$\Delta\text{weight1}_{\text{previous}}$ is the previous amount of change in the first hidden layer weight matrix.

If the relation given above is expanded term by term,

$$\text{weight1}(1,1) = \text{weight1}(1,1) + \eta f(1)\,\text{input}(1) + \theta\,\Delta\text{weight1}_{\text{previous}}(1,1) \quad \textbf{(3.68)}$$

$$\text{weight1}(1,2) = \text{weight1}(1,2) + \eta f(1)\,\text{input}(2) + \theta\,\Delta\text{weight1}_{\text{previous}}(1,2) \quad \textbf{(3.69)}$$

$$\text{weight1}(1,3) = \text{weight1}(1,3) + \eta f(1)\,\text{input}(3) + \theta\,\Delta\text{weight1}_{\text{previous}}(1,3) \quad \textbf{(3.70)}$$

.

.

$$\text{weight1}(1,n\_input) = \text{weight1}(1,n\_input) + \eta f(1)\,\text{input}(n\_input) + \theta\,\Delta\text{weight1}_{\text{previous}}(1,n\_input) \quad \textbf{(3.71)}$$

$$\text{weight1}(2,1) = \text{weight1}(2,1) + \eta f(2)\,\text{input}(1) + \theta\,\Delta\text{weight1}_{\text{previous}}(2,1) \quad \textbf{(3.72)}$$

$$\text{weight1}(2,2) = \text{weight1}(2,2) + \eta f(2)\,\text{input}(2) + \theta\,\Delta\text{weight1}_{\text{previous}}(2,2) \quad \textbf{(3.73)}$$

$$\text{weight1}(2,3) = \text{weight1}(2,3) + \eta f(2)\,\text{input}(3) + \theta\,\Delta\text{weight1}_{\text{previous}}(2,3) \quad \textbf{(3.74)}$$

.

.

$$\text{weight1}(2,n\_input) = \text{weight1}(2,n\_input) + \eta f(2)\,\text{input}(n\_input) + \theta\,\Delta\text{weight1}_{\text{previous}}(2,n\_input) \quad \textbf{(3.75)}$$

.

.

$$weight1(n\_first\_hidden,1) = weight1(n\_first\_hidden,1) + \eta f(n\_first\_hidden)\, input(1) + \theta\, \Delta weight1_{previous}(n\_first\_hidden,1) \quad \textbf{(3.76)}$$

$$weight1(n\_first\_hidden,2) = weight1(n\_first\_hidden,2) + \eta f(n\_first\_hidden)\, input(2) + \theta\, \Delta weight1_{previous}(n\_first\_hidden,2) \quad \textbf{(3.77)}$$

$$weight1(n\_first\_hidden,3) = weight1(n\_first\_hidden,3) + \eta f(n\_first\_hidden)\, input(3) + \theta\, \Delta weight1_{previous}(n\_first\_hidden,3) \quad \textbf{(3.78)}$$

.

.

$$weight1(n\_first\_hidden,n\_input) = weight1(n\_first\_hidden,n\_input) + \eta f(n\_first\_hidden)\, input(n\_input) + \theta\, \Delta weight1_{previous}(n\_first\_hidden,n\_input) \quad \textbf{(3.79)}$$

3.4.8.2 Updating second hidden layer weight matrix

Change in the second hidden layer weight matrix depends on the second hidden layer error, the input to the second hidden layer, the previous amount of change in the second hidden layer weight matrix, the learning rate and the momentum factor. It should be noted that the input to the second hidden layer is the output from the first hidden layer, namely hidden1_log. Table 3.9 gives the values used while updating the second hidden layer weight matrix.

Mathematically, weight2(i,j) is updated as

$$weight2(i,j) = weight2(i,j) + \eta e(i)\, hidden1\_log(j) + \theta\, \Delta weight2_{previous}(i,j) \quad \textbf{(3.80)}$$

$(i = 1, 2, \ldots n\_second\_hidden)$

$(j = 1, 2, \ldots n\_first\_hidden)$

where η is the learning rate,

e is the second hidden layer error,

hidden1_log, output of the first hidden layer, is the input to the second hidden layer,

θ is the momentum factor and

$\Delta weight2_{previous}$ is the previous amount of change in the second hidden layer weight matrix.

Table 3.9: Updating the second hidden layer weight matrix

Name	Description	Size
n_first_hidden	number of neurons in the first hidden layer	1
n_second_hidden	number of neurons in the second hidden layer	1
weight2	second hidden layer weight matrix	(n_second_hidden, n_first_hidden)
e	second hidden layer error	(1, n_second_hidden)
hidden1_log	output of the first hidden layer, input to the second hidden layer	(1, n_first_hidden)
$\Delta weight2_{previous}$	previous amount of change in the second hidden layer weight matrix	(n_second_hidden, n_first_hidden)
η	learning rate	1
θ	momentum factor	1

If relation (3.80) is expanded term by term,

$$weight2(1,1) = weight2(1,1) + \eta\, e(1)\, hidden1\_log(1) + \theta\, \Delta weight2_{previous}(1,1) \quad \textbf{(3.81)}$$

$$weight2(1,2) = weight2(1,2) + \eta\, e(1)\, hidden1\_log(2) + \theta\, \Delta weight2_{previous}(1,2) \quad \textbf{(3.82)}$$

$$weight2(1,3) = weight2(1,3) + \eta\, e(1)\, hidden1\_log(3) + \theta\, \Delta weight2_{previous}(1,3) \quad \textbf{(3.83)}$$

.

.

$$weight2(1,n\_first\_hidden) = weight2(1,n\_first\_hidden) + \eta\, e(1)\, hidden1\_log(n\_first\_hidden) + \theta\, \Delta weight2_{previous}(1,n\_first\_hidden) \quad \textbf{(3.84)}$$

$$weight2(2,1) = weight2(2,1) + \eta\, e(2)\, hidden1\_log(1) + \theta\, \Delta weight2_{previous}(2,1) \quad \textbf{(3.85)}$$

$$weight2(2,2) = weight2(2,2) + \eta\, e(2)\, hidden1\_log(2) + \theta\, \Delta weight2_{previous}(2,2) \quad \textbf{(3.86)}$$

$$weight2(2,3) = weight2(2,3) + \eta\, e(2)\, hidden1\_log(3) + \theta\, \Delta weight2_{previous}(2,3) \quad \textbf{(3.87)}$$

.

.

$$\begin{aligned} weight2(2,n\_first\_hidden) = {} & weight2(2,n\_first\_hidden) \\ & + \eta\, e(2)\, hidden1\_log(n\_first\_hidden) \\ & + \theta\, \Delta weight2_{previous}(2,n\_first\_hidden) \end{aligned} \quad \textbf{(3.88)}$$

.

.

$$\begin{aligned} weight2(n\_second\_hidden,1) = {} & weight2(n\_second\_hidden,1) \\ & + \eta\, e(n\_second\_hidden)\, hidden1\_log(1) \\ & + \theta\, \Delta weight2_{previous}(n\_second\_hidden,1) \end{aligned} \quad \textbf{(3.89)}$$

$$\begin{aligned} weight2(n\_second\_hidden,2) = {} & weight2(n\_second\_hidden,2) \\ & + \eta\, e(n\_second\_hidden)\, hidden1\_log(2) \\ & + \theta\, \Delta weight2_{previous}(n\_second\_hidden,2) \end{aligned} \quad \textbf{(3.90)}$$

$$\begin{aligned} weight2(n\_second\_hidden,3) = {} & weight2(n\_second\_hidden,3) \\ & + \eta\, e(n\_second\_hidden)\, hidden1\_log(3) \\ & + \theta\, \Delta weight2_{previous}(n\_second\_hidden,3) \end{aligned} \quad \textbf{(3.91)}$$

.

.

$$\begin{aligned} & weight2(n\_second\_hidden,n\_first\_hidden) = \\ & \quad weight2(n\_second\_hidden,n\_first\_hidden) \\ & \quad + \eta\, e(n\_second\_hidden)\, hidden1\_log(n\_first\_hidden) \\ & \quad + \theta\, \Delta weight2_{previous}(n\_second\_hidden,n\_first\_hidden) \end{aligned} \quad \textbf{(3.92)}$$

3.4.8.3 Updating output layer weight matrix

Change in the output layer weight matrix depends on the output layer error, the input to the output layer, the previous amount of change in the output layer weight matrix, the learning rate and the momentum factor. It should be noted that the input to the

output layer is the output of the second hidden layer, namely hidden2_log. Table 3.10 gives the values used while updating the output layer weight matrix.

Table 3.10: Updating the output layer weight matrix

Name	Description	Size
n_second_hidden	number of neurons in the second hidden layer	1
n_output	number of neurons in the output layer	1
weight3	output layer weight matrix	(n_output, n_second_hidden)
d	output layer error	(1, n_output)
hidden2_log	output from the second hidden layer	(1, n_second_hidden)
$\Delta weight3_{previous}$	previous amount of change in the output layer weight matrix	(n_output, n_second_hidden)
η	learning rate	1
θ	momentum factor	1

Mathematically, weight3(i,j) is updated as

$$weight3(i,j) = weight3(i,j) + \eta\, d(i)\, hidden2\_log(j) + \theta\, \Delta weight3_{previous}(i,j) \qquad \textbf{(3.93)}$$

$$(i = 1, 2, \ldots n\_output)$$
$$(j = 1, 2, \ldots n\_second\_hidden)$$

where η is the learning rate,

d is the output layer error,

hidden2_log, output of the second hidden layer, is the input to the output layer,

θ is the momentum factor and

$\Delta weight3_{previous}$ is the previous amount of change in the output layer weight matrix.

If the relation given above is expanded term by term,

$$weight3(1,1) = weight3(1,1) + \eta\, d(1)\, hidden2\_log(1) + \theta\, \Delta weight3_{previous}(1,1) \qquad \textbf{(3.94)}$$

$$weight3(1,2) = weight3(1,2) + \eta\, d(1)\, hidden2\_log(2) + \theta\, \Delta weight3_{previous}(1,2) \quad \textbf{(3.95)}$$

$$weight3(1,3) = weight3(1,3) + \eta\, d(1)\, hidden2\_log(3) + \theta\, \Delta weight3_{previous}(1,3) \quad \textbf{(3.96)}$$

.

.

$$\begin{aligned} weight3(1,n\_second\_hidden) = {} & weight3(1,n\_second\_hidden) \\ & + \eta d(1)\, hidden2\_log(n\_second\_hidden) \\ & + \theta\, \Delta weight3_{previous}(1,n\_second\_hidden) \end{aligned} \quad \textbf{(3.97)}$$

$$weight3(2,1) = weight3(2,1) + \eta\, d(2)\, hidden2\_log(1) + \theta\, \Delta weight3_{previous}(2,1) \quad \textbf{(3.98)}$$

$$weight3(2,2) = weight3(2,2) + \eta\, d(2)\, hidden2\_log(2) + \theta\, \Delta weight3_{previous}(2,2) \quad \textbf{(3.99)}$$

$$weight3(2,3) = weight3(2,3) + \eta\, d(2)\, hidden2\_log(3) + \theta\, \Delta weight3_{previous}(2,3) \quad \textbf{(3.100)}$$

.

.

$$\begin{aligned} weight2(2,n\_second\_hidden) = {} & weight3(2,n\_second\_hidden) \\ & + \eta\, d(2)\, hidden2\_log(n\_second\_hidden) \\ & + \theta\, \Delta weight3_{previous}(2,n\_second\_hidden) \end{aligned} \quad \textbf{(3.101)}$$

.

.

$$\begin{aligned} weight3(n\_output,1) = {} & weight3(n\_output,1) \\ & + \eta\, d(n\_output)\, hidden2\_log(1) \\ & + \theta\, \Delta weight3_{previous}(n\_output,1) \end{aligned} \quad \textbf{(3.102)}$$

$$\begin{aligned} weight3(n\_output,2) = {} & weight3(n\_output,2) \\ & + \eta\, d(n\_output)\, hidden2\_log(2) \\ & + \theta\, \Delta weight3_{previous}(n\_output,2) \end{aligned} \quad \textbf{(3.103)}$$

$$weight3(n\_output, 3) = weight3(n\_output, 3) + \eta\, d(n\_output)\, hidden2\_log(3) + \theta\, \Delta weight3_{previous}(n\_output, 3) \quad \textbf{(3.104)}$$

.

$$weight3(n\_output, n\_second\_hidden) = weight3(n\_output, n\_second\_hidden) + \eta\, d(n\_output)\, hidden2\_log(n\_second\_hidden) + \theta\, \Delta weight3_{previous}(n\_output, n\_second\_hidden) \quad \textbf{(3.105)}$$

3.4.8.4 Updating first hidden layer bias

Change in the first hidden layer bias depends on the first hidden layer error, the previous amount of change in the first hidden layer bias, the learning rate and the momentum factor. Table 3.11 gives the values used while updating the first hidden layer bias.

Table 3.11: Updating the first hidden layer bias

Name	Description	Size
n_first_hidden	number of neurons in the first hidden layer	1
bias1	first hidden layer bias	(1, n_first_hidden)
f	first hidden layer error	(1, n_first_hidden)
$\Delta bias1_{previous}$	previous amount of change in the first hidden layer bias	(1, n_first_hidden)
η	learning rate	1
θ	momentum factor	1

Mathematically, bias1(1,i) is updated as

$$bias1(1,i) = bias1(1,i) + \eta f(1,i) + \theta\, \Delta bias1_{previous}(1,i) \quad (i=1, 2, \ldots n\_first\_hidden) \quad \textbf{(3.106)}$$

where η is the learning rate,

f is the first hidden layer error,

θ is the momentum factor and

$\Delta bias1_{previous}$ is the previous amount of change in the first hidden layer bias.

If the relation is expanded term by term,

$$bias1(1,1) = bias1(1,1) + \eta f(1,1) + \theta \Delta bias1_{previous}(1,1) \quad \textbf{(3.107)}$$

$$bias1(1,2) = bias1(1,2) + \eta f(1,2) + \theta \Delta bias1_{previous}(1,2) \quad \textbf{(3.108)}$$

$$bias1(1,3) = bias1(1,3) + \eta f(1,3) + \theta \Delta bias1_{previous}(1,3) \quad \textbf{(3.109)}$$

.

.

$$bias1(1,n\_first\_hidden) = bias1(1,n\_first\_hidden) + \eta f(1,n\_first\_hidden) + \theta \Delta bias1_{previous}(1,n\_first\_hidden) \quad \textbf{(3.110)}$$

3.4.8.5 Updating second hidden layer bias

Change in the second hidden layer bias depends on the second hidden layer error, the previous amount of change in the second hidden layer bias, the learning rate and the momentum factor. Table 3.12 gives the values used while updating the second hidden layer bias.

Table 3.12: Updating the second hidden layer bias

Name	Description	Size
n_second_hidden	number of neurons in the second hidden layer	1
bias2	second hidden layer bias	(1, n_second_hidden)
e	second hidden layer error	(1, n_second_hidden)
$\Delta bias2_{previous}$	previous amount of change in the second hidden layer bias	(1, n_second_hidden)
η	learning rate	1
θ	momentum factor	1

Mathematically, bias2(1,i) is updated as

$$bias2(1,i) = bias2(1,i) + \eta e(1,i) + \theta \Delta bias2_{previous}(1,i) \quad \textbf{(3.111)}$$
$$(i = 1, 2, \ldots n\_second\_hidden)$$

where η is the learning rate,

e is the second hidden layer error,

θ is the momentum factor and

$\Delta bias2_{previous}$ is the previous amount of change in the second hidden layer bias. If the relation is expanded term by term,

$$bias2(1,1) = bias2(1,1) + \eta e(1,1) + \theta \Delta bias2_{previous}(1,1) \quad \textbf{(3.112)}$$

$$bias2(1,2) = bias2(1,2) + \eta e(1,2) + \theta \Delta bias2_{previous}(1,2) \quad \textbf{(3.113)}$$

$$bias2(1,3) = bias2(1,3) + \eta e(1,3) + \theta \Delta bias2_{previous}(1,3) \quad \textbf{(3.114)}$$

.

.

$$bias2(1,n\_second\_hidden) = bias2(1,n\_second\_hidden) + \eta e(1,n\_second\_hidden) + \theta \Delta bias2_{previous}(1,n\_second\_hidden) \quad \textbf{(3.115)}$$

3.4.8.6 Updating output layer bias

Change in the output layer bias depends on the output layer error, the previous amount of change in the output layer bias, the learning rate and the momentum factor. Table 3.13 gives the values used while updating the output layer bias.

Table 3.13: Updating the output layer bias

Name	Description	Size
n_output	number of neurons in the output layer	1
bias3	output layer bias	(1,n_output)
d	output layer error	(1,n_output)
$\Delta bias3_{previous}$	previous amount of change in the output layer bias	(1,n_output)
η	learning rate	1
θ	momentum factor	1

Mathematically, bias3(1,i) is updated as

$$bias3(1,i) = bias3(1,i) + \eta d(1,i) + \theta \Delta bias3_{previous}(1,i) \quad \textbf{(3.116)}$$
$$(i=1,2,..n\_output)$$

where η is the learning rate,

d is the output layer error,

θ is the momentum factor and

$\Delta bias3_{previous}$ is the previous amount of change in the output layer bias.

If the relation is expanded term by term,

$$bias3(1,1) = bias3(1,1) + \eta\, d(1,1) + \theta\, \Delta bias3_{previous}(1,1) \quad \textbf{(3.117)}$$

$$bias3(1,2) = bias3(1,2) + \eta\, d(1,2) + \theta\, \Delta bias3_{previous}(1,2) \quad \textbf{(3.118)}$$

$$bias3(1,3) = bias3(1,3) + \eta\, d(1,3) + \theta\, \Delta bias3_{previous}(1,3) \quad \textbf{(3.119)}$$

.

.

$$bias3(1,n\_output) = bias3(1,n\_output) + \eta\, d(1,n\_output) + \theta\, \Delta bias3_{previous}(1,n\_output) \quad \textbf{(3.120)}$$

3.4.9 Total error

Repeating steps 3.4.1 – 3.4.8.6 until all training data are included in the computations is called one epoch. These steps can be repeated as many times as possible until the total error of the network falls below an acceptable level. Total error is the way to monitor how well the NN performs. A sum squared error or an absolute error may be used for this purpose. Since the aim of the NN is to learn to produce the target values in the training set once it has been provided with the input values in the training set, the error can simply be the difference between the output of the NN and the target values that it is supposed to produce. This error is calculated across all training patterns by using the updated weights and biases at the end of each epoch. Thus, the mathematical measure of the performance of the NN, in the case of calculating an absolute error, is

$$\text{Total error} = \sum_{i=1}^{\text{number of data points}} \sum_{j=1}^{\text{number of output neurons}} (output(j) - target(j)) \quad \textbf{(3.121)}$$

where output(j) is the output of the j^{th} neuron in the output layer and

target.(j) is the target value of the j^{th} neuron in the data pattern.

The flowchart given in Figure 3.10 summarizes how this backpropagation algorithm works.

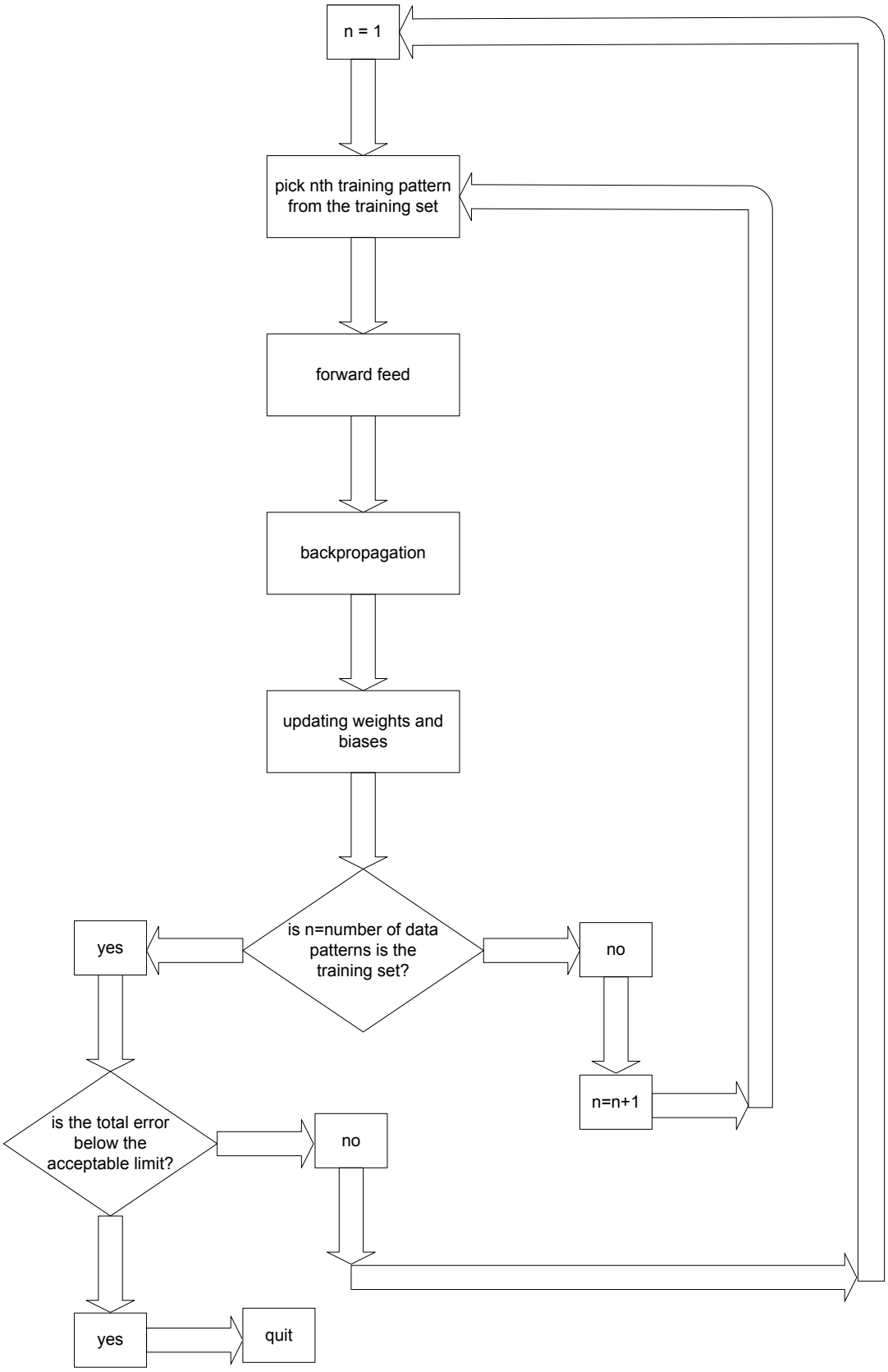


Figure 3.10: Summary of the backpropagation algorithm

A quick examination of the algorithm reveals that the backpropagation network trains itself to recognize features in the input patterns and slowly changes the network connection weights and biases to minimize the error across all training data patterns. Backpropagation might require extremely long training times but it must be kept in mind that training a NN for a particular application will only be done once. After the correct weights and biases are obtained, only the feedforward part of the NN will be used in order to obtain the right output values for an input pattern. NN will simply produce the right outputs for an input data pattern.

3.5 Fine Points to Improve the Performance of the Backpropagation Algorithm

The following points might be helpful in improving the performance of the backpropagation algorithm

1) It might be helpful to slowly decrease the learning rate, ultimately asymptotically to 0. It is easy to add to the algorithm and is helps in converging to a small total error.

2) Momentum term can be extremely helpful in helping neural networks to train faster. Momentum term is a measure of the effect of the previous change in the weights and biases on the current amount of change.

3) Epoch-based training can be carried out, where the weights and biases are not updated after the presentation of a new input data pattern to the network, but after the end of each epoch. In the case of epoch-based encoding, weight and bias adjustments are done based on the cumulative error values at the end of each epoch.

4) The absolute error optimization criterion might be replaced with another type of error criterion.

5) Having too many neurons in the hidden layers does not always improve the performance of the NN, thus the optimal number of neurons in the hidden layers must be found by trial-and-error.

4. ADAPTATION OF NEURAL NETWORKS TO THE FORWARD KINEMATICS PROBLEM OF PARALLEL MECHANISMS

The type of the NN used in this application is the feedforward network with two hidden layers and is trained using backpropagation algorithm. The reason why it is useful to use neural networks in solving the forward kinematics of the 6-3 SPM is that the conventional method of solving the forward kinematics problem is to solve a 16^{th} order polynomial equation and obtain 16 candidate results. These results are then tested to find the real physical solution. Out of the 16 possible solutions, only one is the real solution. The real solution is found by testing the 16 results to see which one satisfies mechanical and geometrical constraints. These constraints are limitations on leg lengths, limitations on passive joint angles and interference of the legs. In this tedious process, infeasible solutions are eliminated to yield the real solution to the physical problem. It is much simpler to train a neural network for this problem. Once a neural network is trained for a particular problem and the right connection weights and biases are obtained and saved, it takes the network milliseconds to produce the output value for a given input set of data patterns. In this case, this input data pattern is the set of leg lengths and the output data pattern is the set of coordinates and rotation angles defining the position and orientation of the top platform of the SPM. If the network is trained so that its outputs have the desired accuracy, there is no need to use a gyroscope or a position sensor to determine the position or orientation of the moving platform of the SPM.

The NN application given in this thesis has six neurons in the input layer and six neurons in the output layer, due to the nature of the FK problem of the 6-3 SPM. Since the FK problem solves for six values, which are the coordinates of the center of gravity of the moving platform of the SPM and the rotation of the moving platform with respect to three axes when six leg lengths are given, it means that the NN needs to produce six outputs in the presence of six inputs. In this case, both n_input and n_output equal six. Since the mechanism has six legs, the input layer must have six neurons. The position of the center of gravity can be represented by

three coordinates and any orientation of the top platform can be given as the combination of three rotations, thus the output layer must also have six neurons. Thus, the NN architecture has six neurons in both the input and the output layers.

Each of the neurons in the input layer represents one leg length since the input layer corresponds to $[l_1, l_2, l_3, l_4, l_5, l_6]$ leg lengths matrix. Each of the neurons in the output layer represents either the position of the center of gravity of the top platform or the orientation of the top platform with respect to the coordinate axes on the base since the output layer corresponds to [x, y, z, γ, β, α] target position and orientation matrix.

It has been observed with trial-and-error that a NN with two hidden layers has better performance than a NN with a single hidden layer. It has further been observed by trial and error that the best NN configuration is one having 10 neurons in the first hidden layer and 12 neurons in the second hidden layer. The activation function f(net) used in the hidden layers is the tangent-sigmoid function, the mathematical equivalent of the tangent-hyperbolic function, defined as

$$f(x) = \frac{2}{1+e^{-2x}} - 1 \qquad \textbf{(4.1)}$$

Backpropagation algorithm is used for the training of the NN. Since trial-and-error is the only way to find the best configuration of the NN to be used for a particular application, many trials have been carried out to find the optimum number of neurons in each hidden layer. It has also been found out that gradually decreasing the learning rate to 0 is helpful for convergence. The measure of performance of the NN is an error function defined as

$$\text{Error} = \sum_{p=1}^{\text{number of data points}} \sum_{k=1}^{\text{number of output neurons}} (\text{output}(1,k) - \text{target}(1,k)) \qquad \textbf{(4.2)}$$
$$(\text{for } k = 1{:}\ 6)$$

where output matrix is the value produced by the NN and target is the desired value that should actually be produced. (output(1,k) - (target (1,k)) is the difference between the output value of the k^{th} neuron in the output layer and the target value of the k^{th} neuron in the output layer. The performance of the NN is further improved by applying the loop method developed in [78]. It should be noted that this error is in radians for the purely rotational case, in meters in the purely translational case. In the

general case, there is no common unit by which both translation and rotation are represented simultaneously. This error function is the way to monitor the performance of the NN, thus the training of the NN stops only when the average error per data pattern, which is obtained by simply dividing the error function defined above by the number of data patterns used in training, drops below an acceptable value. After the training of the NN is complete, it is tested with other data sets to see whether it can produce accurate results for data patterns that were not in the training set.

4.1 Architecture of the SPM Used

The SPM model used in this application has a hexagonal base and a triangular top platform. The coordinate system fixed to the base has its origin at the center of the hexagon. The x, y, and z coordinates of the vertices of the base hexagon, in meters, are as the following:

$B_1 = [0.57693, -0.338, 0]$ **(4.3)**

$B_2 = [0.57693, 0.338, 0]$ **(4.4)**

$B_3 = [0.00425, 0.66863, 0]$ **(4.5)**

$B_4 = [-0.58118, 0.33063, 0]$ **(4.6)**

$B_5 = [-0.58118, -0.33063, 0]$ **(4.7)**

$B_6 = [0.00425, -0.66863, 0]$ **(4.8)**

It can easily be seen that the side lengths of neighboring sides of the base platform alternate between 0.6760 and 0.6613 meters where two consecutive sides have unequal lengths. In simpler terms, sides B_1B_2, B_3B_4 and B_5B_6 have length 0.6760 meters and sides B_2B_3, B_4B_5 and B_1B_6 have length 0.6613 meters. The moving platform is an equailateral triangle with side length 0.9963 meters.

4.2 Generation of Training Data

In order to generate the data to be used while training the neural network, a simple mathematical procedure is used. It must be noted that since the NN will be trained to solve the forward kinematics of the Stewart Platform Mechanism, the input values to be used during NN training are the lengths of the six legs of the mechanism and the target values are the six values indicating the position of the center of gravity of the moving platform with respect to the coordinate system at the base and orientation of the moving platform of the Stewart Platform Mechanism (SPM) with respect to the base. Both the input values and the target values have the form of row matrices of size 1*6. Each leg length constitutes one element of the input matrix. Coordinates of the position of the top platform of the SPM (x, y, z) constitutes the first three elements of the target matrix while the orientation of the top platform with respect to three coordinate axes (γ, β, α) constitutes the last three elements of the target matrix. In terms of the NN, each leg length corresponds to one neuron in the input layer and each of the values indicating the position and orientation of the moving platform corresponds to one neuron in the output layer.

In short, inverse kinematics equations are used to generate the data sets to be used while training the neural network. Basic steps of the data generation process are the following:

1) Determination of the three angle bisector vectors in nominal form (without any rotations) in terms of the side lengths of the top platform of the SPM (Figure 4.1)
2) a. In the case of purely rotational or purely translational data: Scanning the pre-determined workspace with the desired step sizes
   b. In the case of general spatial data where translation and rotation are combined: Random generation of target vectors within the allowed limits
3) Determination of the rotation matrices
4) Determination of the angle bisector vectors after the rotations
5) Determination of the coordinates of the vertices of the top platform of the SPM
6) Solving the inverse kinematics equations to obtain leg lengths

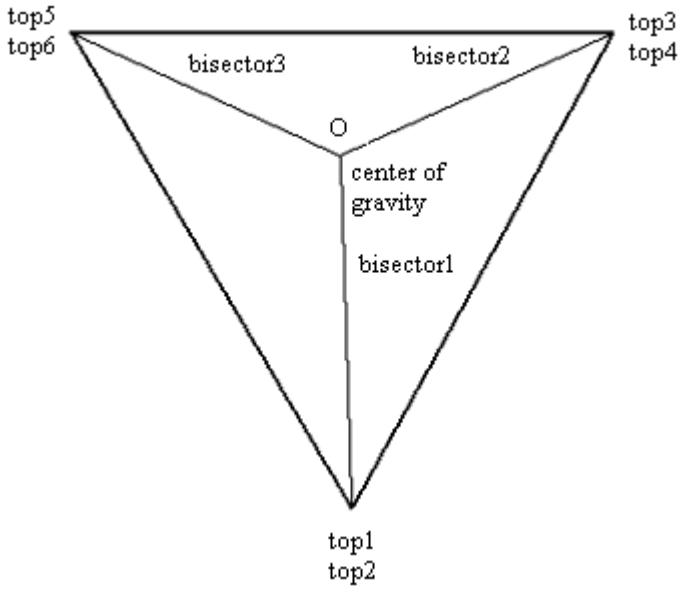


Figure 4.1: Moving platform of the 6-3 SPM

In step 1, the angle bisector vectors are given in terms of the side lengths of the top platform. It must be noted that these vectors are given with respect to the coordinate system fixed to the center of gravity of the top platform. Since the top platform is an equilateral triangle, geometric manipulations easily reveal the coordinates of the vertices of the top platform because the angle bisector vectors originate at the center of gravity of the top platform, which is the origin of the coordinate system fixed to the top platform, and end at the vertices.

Step 2 is where the generation of the vectors describing the position of the center of gravity of the top platform and orientation of the top platform of the SPM takes place. In this process, for the purely rotational and purely translational cases, the workspace is scanned within certain upper and lower limits. Due to mechanical and geometrical limitations such as limitations on leg lengths, limitations on twist angles of passive joints and interference of the legs, the top platform of the SPM can only operate in a limited workspace. In other words, the center of gravity of the top platform of the SPM can only sweep a limited space and the top platform can only make limited rotations. The x coordinate of the center of gravity of the top platform can only take values within the range [xmin, xmax], the y coordinate of the center of gravity of the top platform can only take values within the range [ymin, ymax], the z coordinate of the center of gravity of the top platform can only take values within the range [zmin, zmax], the rotation around the x axis can only be within the range [gammamin, gammamax], the rotation around the y axis can only be within the range

[betamin, betamax] and the rotation around the z axis can only be within the range [alfamin, alfamax].

Scanning procedure of the workspace is conducted as the following: In the purely rotational case, the center of gravity of the top platform is taken to a point (x, y, z) and then the top platform is allowed to go through certain rotations with respect to three axes. The point where the center of gravity of the top platform is fixed is the midpoint of the heights which give the shortest and longest leg lengths when the platform moves only in the z direction. This is because this ''center'' point allows more rotations than any other point within the workspace of the 6-3 SPM. At this point, inverse kinematics equations are solved and leg lengths are checked to see whether they satisfy mechanical constraints. If the leg lengths are within the allowed upper and lower limits, then the data pattern is added to the training set of the NN. In the purely translational case, the center of gravity of the moving platform scans the pre-determined workspace by moving in the x, y, and z directions with given stepsizes. In this case, it is assumed that the top platform remains parallel to the base. At this point, inverse kinematics equations are solved and leg lengths are checked to see whether they satisfy mechanical constraints. If the leg lengths are within the allowed upper and lower limits, then the data pattern is added to the training set of the NN. In the general data set where both translation and rotation take place, target values are randomly generated, rather than by scanning the workspace in 6 directions. A random number generator generates numbers in the range [0, 1], and then these numbers are scaled so that their ranges are given by the upper and lower limits given in step 1. This means that if the generator generates 0, it will be scaled to the lower bound of that variable, and if the generator generates 1, it will be scaled to the upper bound of that variable. In mathematical terminology,

the x coordinate of the center of gravity of the top platform will be transformed according to

$$x = (x_{max} - x_{min})\,(\text{random number}) + x_{min} \qquad \textbf{(4.9)}$$

where x_{max} is the upper limit of the x coordinate and x_{min} is the lower limit of the x coordinate of the center of gravity of the top platform,

the y coordinate of the center of gravity of the top platform will be transformed according to

$$y = (y_{max} - y_{min})\,(\text{random number}) + y_{min} \quad \textbf{(4.10)}$$

where y_{max} is the upper limit of the y coordinate and y_{min} is the lower limit of the y coordinate of the center of gravity of the top platform,

the z coordinate of the center of gravity of the top platform will be transformed according to

$$z = (z_{max} - z_{min})\,(\text{random number}) + z_{min} \quad \textbf{(4.11)}$$

where z_{max} is the upper limit of the z coordinate and z_{min} is the lower limit of the z coordinate of the center of gravity of the top platform,

the rotation of the top platform with respect to the x axis, γ, will be transformed according to

$$\gamma = (\gamma_{max} - \gamma_{min})\,(\text{random number}) + \gamma_{min} \quad \textbf{(4.12)}$$

where γ_{max} is the upper bound of the rotation about the x axis and γ_{min} is the lower bound of the rotation about the x axis,

the rotation of the top platform with respect to the y axis, β, will be transformed according to

$$\beta = (\beta_{max} - \beta_{min})\,(\text{random number}) + \beta_{min} \quad \textbf{(4.13)}$$

where β_{max} is the upper bound of the rotation about the y axis and β_{min} is the lower bound of the rotation about the y axis,

the rotation of the top platform with respect to the z axis, α, will be transformed according to

$$\alpha = (\alpha_{max} - \alpha_{min})\,(\text{random number}) + \alpha_{min} \quad \textbf{(4.14)}$$

where α_{max} is the upper bound of the rotation about the z axis and α_{min} is the lower bound of the rotation about the z axis.

Here x, y, and z describe the position of the center of gravity of the top platform, and γ, β, and α describe the angles of rotation of the top platform with respect to the x, y, and z axes. It should be noted that the random numbers given in the six equations

above are unequal. Once again, inverse kinematics equations are solved and leg lengths are checked to see whether they satisfy mechanical constraints. If the leg lengths are within the allowed upper and lower limits, then the data pattern is added to the training set of the NN. During this step of the data generation process, feasible data are saved as the training data. Both the input and the target data patterns are in the form of 1*6 matrices. Input data patterns describe the leg lengths (l_1, l_2, l_3, l_4, l_5, and l_6) while the target data patterns describe the position and orientation of the top platform where the first three elements (x, y, z) describe the position of the center of gravity of the top platform with respect to the coordinate system at the base platform of the SPM and the last three (γ, β, α) describe the orientation of the top platform with respect to the three coordinate axes at the base platform of the SPM. These data patterns are saved so that they can be accessed later during the training of the NN and used as the input and target values.

In the 3rd step of the data generation process, rotation matrices are calculated. Since the rotation matrices are functions of the rotation angles, the three angles generated in step 2 are substituted to obtain three rotation matrices for a target vector. Physical meaning of rotation matrices and the mathematical relations used to obtain them are given in detail in section 4.3. The physical meaning behind this step of the data generation process is that it is assumed that the top platform of the SPM reaches a certain orientation through three consecutive rotations about the x, y and z axes, respectively. In order to solve the inverse kinematics equations and obtain the lengths of the legs, the coordinates of the vertices of the top platform need to be known, because a leg length is the norm of the difference between the coordinate of the point where a leg is attached to the top platform and the coordinate of the point where it is attached to the base. Coordinates of each vertex can be found by adding the related bisector vector to the coordinates of the center of gravity of the top platform of the SPM. Coordinates of the vertices of the base platform are given in section 4.1 with respect to the coordinate system fixed to the base. Therefore, in order to be able to calculate leg lengths, the coordinates of the vertices of the top platform must also be given with respect to the fixed coordinate system. Since the coordinates of the vertices can be found by adding the related bisector vector to the coordinates of the center of gravity of the top platform with respect to the fixed reference frame, the bisector vectors must also be expressed with respect to the fixed coordinate system.

In step 1, these angle bisectors were given in coordinates of the reference system on the top platform. What is done in this step is to multiply them by the appropriate rotation matrix so that they are expressed in coordinates of the reference system fixed to the base. It was stated above that the top platform is assumed to undergo three consecutive rotations about the x, y and z axes, respectively, in order to attain its orientation. To obtain the R_x, R_y, and R_z rotation matrices, where R_x is the rotation matrix about x axis, R_y is the rotation matrix about y axis, and R_z is the rotation matrix about z axis, gamma, beta and alpha values generated in step 2 are used and substituted into

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix} \quad \textbf{(4.15)}$$

$$R_y = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \quad \textbf{(4.16)}$$

$$R_z = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \textbf{(4.17)}$$

Since the aim here is to convert the coordinates given in terms of the reference system fixed to the top platform into the coordinates given with respect to the reference system fixed to the base platform, the problem can be thought of as moving from the body-fixed reference frame to the space-fixed reference frame. It was stated above that if a point is given in coordinates of the body-fixed coordinate frame and it is desired to express it in coordinates of the space-fixed coordinate frame, the matrix that defines the rotations will be obtained by multiplying the rotation matrices in the reverse order of the rotations. Thus, the rotation matrix can be given as $R = R_z\ R_y\ R_x$. Details on rotation matrices are given in section 4.3.

Step 4 is simply multiplying the bisector vectors in coordinates of the body-fixed reference system with the rotation matrix found in step 3 so that the bisector vectors are expressed in coordinates of the reference system fixed to the base. That is,

$$bisector_1^{\,1} = R\,bisector_1 \quad (4.18)$$

$$bisector_2^{\,1} = R\,bisector_2 \quad (4.19)$$

$$bisector_3^{\,1} = R\,bisector_3 \quad (4.20)$$

where $bisector_i$ (i=1, 2, 3) represents the angle bisectors in coordinates of the reference frame at the top platform (body-fixed reference frame) and $bisector_i^{\,1}$ (i=1, 2, 3) represents the angle bisectors in coordinates of the reference frame at the base platform(space-fixed reference frame).

In step 5, the bisector vectors found in step 4 are added to the coordinates of the center of gravity of the top platform so that the vertices of the top platform are obtained in coordinates of the reference system fixed to the base. If the first three elements of the target matrix, meaning the coordinates of the center of gravity of the top platform, namely (x, y, z), are represented in a 1*3 matrix called cg, abbreviation for center of gravity, the coordinates of the vertices of the top platform can be given by

$$vertex_1 = bisector_1^{\,1} + cg \quad (4.21)$$

$$vertex_2 = bisector_2^{\,1} + cg \quad (4.22)$$

$$vertex_3 = bisector_3^{\,1} + cg \quad (4.23)$$

Finally, in step 6, leg lengths are obtained by taking the norms of the vectors obtained by subtracting the coordinates of the point where a leg is attached to the base from the coordinates of the point where a leg is attached to the top. It must be noted that since the mechanism is a 6-3 SPM, each leg has its own point of attachment to the base platform while two consecutive legs share a common point of attachment to the top platform. If the six vertices of the base platform are represented as $base_1$, $base_2$, $base_3$, $base_4$, $base_5$ and $base_6$, the leg lengths are found as

$$length_1 = norm(vertex_1 - base_1) \quad (4.24)$$

$$length_2 = norm(vertex_1 - base_2) \quad (4.25)$$

$$length_3 = norm(vertex_2 - base_3) \quad (4.26)$$

$$length_4 = norm(vertex_2 - base_4) \quad (4.27)$$

$$length_5 = norm(vertex_3 - base_5) \quad (4.28)$$

$$length_6 = norm(vertex_3 - base_6) \quad (4.29)$$

where $length_i$ $(i=1, 2, .. 6)$ is the length of the ith leg.

If these leg lengths are feasible values, then they are stored as elements of 1*6 matrices and saved in a file as the input data, and the coordinates of the center of gravity of the top platform and orientation of the top platform are stores as elements of 1*6 matrices and saved in a file as the output data to be accessed and used in NN training.

Thus, three types of data have been generated using this procedure. The first one is purely rotational data where the center of gravity of the top platform of the SPM is fixed to a point in space and the top platform undergoes rotations with respect to the three coordinate axes, the second one is purely translational data where the top platform of the SPM is only translated in three directions parallel to the base platform and the third one is general spatial data where the top platform of the SPM is both translated and rotated. In the first data type, only the orientation of the top platform changes, in the second data type, only the position of the top platform changes, and in the third type, both the position and the orientation of the top platform change. The entire procedure is the same in all three cases except for a small difference in step 2 where the generation of target matrices takes place.

In the case of purely rotational data, the center of gravity of the moving platform stays fixed at a point and the top platform undergoes rotations with respect to the x, y, and z axes while its center of gravity is fixed. Thus, in this case, the first three elements of the target data patterns are the same in all data patterns. In the case of purely translational data, the top platform only goes through translational motion along the three axes but does not rotate. In other words, the top platform moves parallel to the base. This means that in this case, the last three elements of all the data patterns in the target data set are zero, indicating the upper platform does not go through any rotations with respect to the base since γ, β, and α indicate the amounts of rotations of the top platform with respect to the base. In the case of general spatial

data, the upper platform is both translated in three directions and rotated with respect to three axes. Figure 4.2 summarizes the key points of the data generation process.

4.2.1 Purely rotational data

Training data have been generated by fixing the center of gravity of the moving platform to the point [0 0 0.8] (meters) and scanning the workspace with stepsizes of 1° where γ is the angle of rotation with respect to the x axis, β is the angle of rotation with respect to the y axis, and α is the angle of rotation with respect to the z axis. γ scans values within the range $\pm 20°$, β scans values within the range $\pm 22°$, α scans values within the range $\pm 24°$ since these are the limits of the rotational workspace about the point [0 0 0.8] (meters). Out of the 90 405 data patterns scanned, 17 123 data fullfilled leg length constraints and were picked as training data.

Two other data sets have been generated for testing the NN by scanning the same workspace with stepsizes of 0.5° and 0.25°. When the workspace whose limits are given above is scanned in 3 directions with stepsizes of 0.5°, 699 273 data patterns were scanned and 136 893 of them satisfied leg length constraints. When the workspace whose limits are given above was scanned in 3 directions with stepsizes of 0.25°, 5 499 921 data patterns were scanned and 1 093 886 of them satisfied leg length constraints. These two data sets were utilized in testing the NN and the loop method.

4.2.2 Purely translational data

Training data have been generated by scanning the workspace in the x, y, and z directions with stepsizes of 10 mm while keeping the top platform parallel to the base. x direction has been scanned within the range ± 450 mm, y direction has been scanned within the range ± 390 mm and z direction has been scanned within the range (600 mm, 930 mm) since these are the limits of the translational workspace of the mechanism. Out of the 244 426 data patterns scanned, 64 803 data fullfilled leg length constraints and were picked as training data.

Two other data sets have been generated for testing the NN by scanning the same workspace in 3 directions with stepsizes of 5 mm and 2.5 mm. When the workspace

whose limits are given above is scanned in 3 directions with step sizes of 5 mm, 1 903 939 data patterns were scanned and 516 811 of them satisfied leg length constraints.

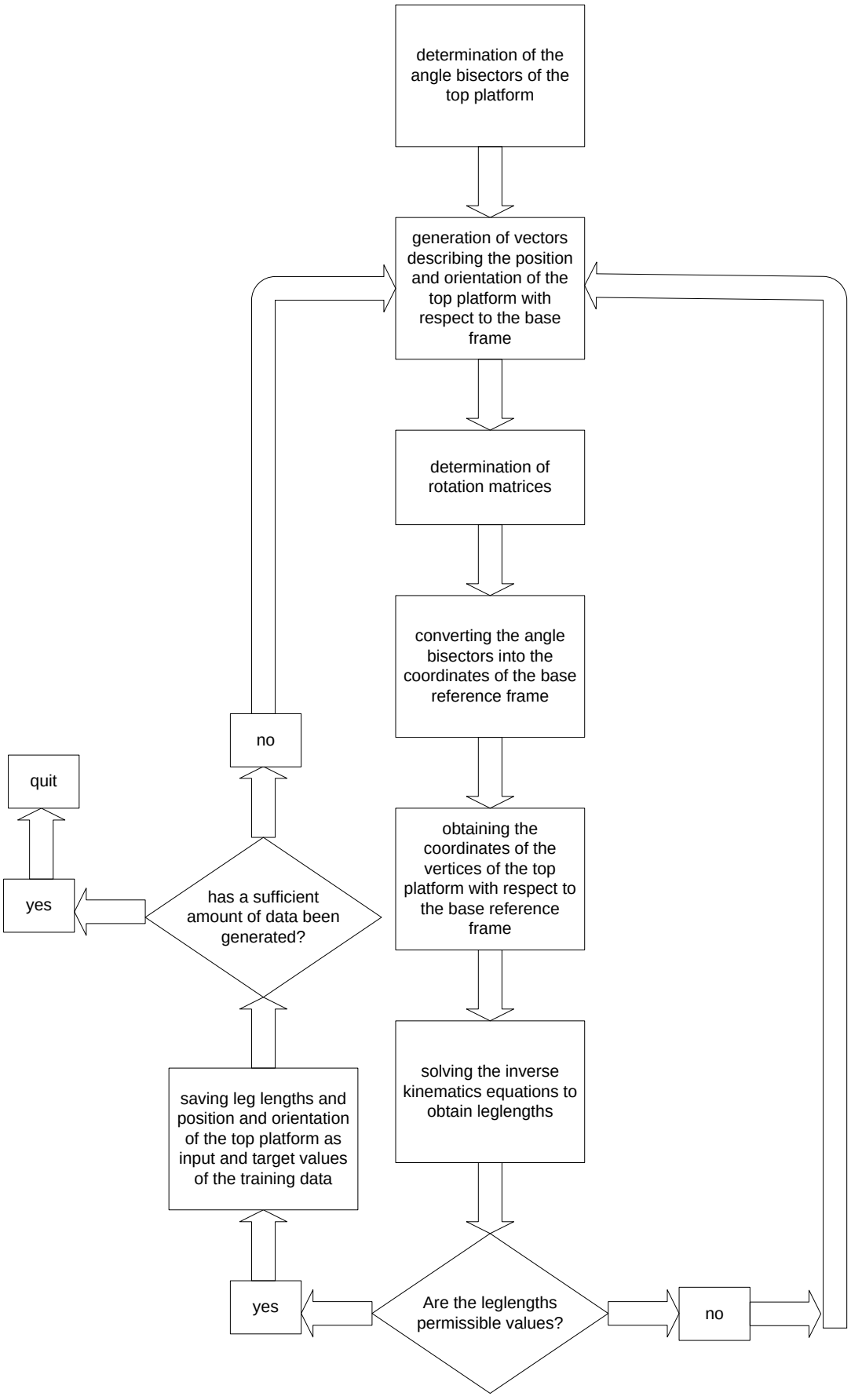


Figure 4.2: Data generation process

When the workspace whose limits are given above was scanned in 3 directions with stepsizes of 2.5 mm, 15 028 069 data patterns were scanned and 4 130 586 of them satisfied leg length constraints. These two data sets were utilized in testing the NN and the loop method.

4.2.3 General data

Training data have been generated by randomly generating values within the ranges $\pm$ 200 mm in the x direction, $\pm$ 200 mm in the y direction, (750 mm, 850 mm) in the z direction, and $\pm$ 10 $\pm$ in the γ, β, and α directions. 53 351 data patterns have been generated and used in training the NN. To test the NN, another set of 106 443 data patterns have been generated.

4.3 Rotation Matrices

Tractable analysis of coordinate systems used and transformations among them are quite important since kinematics analysis is dependent entirely upon them. Consider two coordinate systems, (x_1, x_2, x_3) and (x_1^1, x_2^1, x_3^1) which are initially coincident. Frame (x_1^1, x_2^1, x_3^1) is rotating about frame (x_1, x_2, x_3) which is fixed in space. There must be a matrix of rotation defining this rotation such that when a vector defining a point in the fixed frame (x_1, x_2, x_3) is multiplied by this matrix of rotation, the result gives the coordinates of the same point in terms of the coordinate system obtained after the rotation, namely (x_1^1, x_2^1, x_3^1). Mathematically,

$$[x_1^1 \; x_2^1 \; x_3^1]^T = \lambda [x_1 \; x_2 \; x_3]^T \qquad \textbf{(4.30)}$$

where λ is the rotation matrix,

$[x_1 \; x_2 \; x_3]^T$ is a point in the fixed coordinate system about which rotations take place and

$[x_1^1 \; x_2^1 \; x_3^1]^T$ gives the coordinates of the same point with respect to the reference system resulting after the rotation.

If a frame undergoes two consecutive rotations, then there will be two matrices of rotation, λ_1 and λ_2 and

$$[x_1^{11} \; x_2^{11} \; x_3^{11}]^T = \lambda_2 [x_1^1 \; x_2^1 \; x_3^1]^T = \lambda_2 \; \lambda_1 [x_1 \; x_2 \; x_3]^T \qquad \textbf{(4.31)}$$

where λ_1 is the rotation matrix of the first rotation,

λ_2 is the rotation matrix of the second rotation,

$[x_1\ x_2\ x_3]^T$ is a point in the coordinate system about which rotations take place,

$[x_1^1\ x_2^1\ x_3^1]^T$ gives the coordinates of the same point with respect to the reference system resulting after the first rotation and

$[x_1^{11}\ x_2^{11}\ x_3^{11}]^T$ gives the coordinates of the same point with respect to the reference system resulting after the second rotation.

Similarly, if a frame undergoes three consecutive rotations, then there will be three matrices of rotation, λ_1, λ_2 and λ_3

$$[x_1^{111}\ x_2^{111}\ x_3^{111}]^T = \lambda_3\ [x_1^{11}\ x_2^{11}\ x_3^{11}]^T = \lambda_3\ \lambda_2\ [x_1^1\ x_2^1\ x_3^1]^T = \lambda_3\ \lambda_2\ \lambda_1\ [x_1\ x_2\ x_3]^T \quad \textbf{(4.32)}$$

where λ_1 is the rotation matrix of the first rotation,

λ_2 is the rotation matrix of the second rotation,

λ_3 is the rotation matrix of the second rotation,

$[x_1\ x_2\ x_3]^T$ is a point in the coordinate system about which rotations take place,

$[x_1^1\ x_2^1\ x_3^1]^T$ gives the coordinates of the same point with respect to the reference system resulting after the first rotation,

$[x_1^{11}\ x_2^{11}\ x_3^{11}]^T$ gives the coordinates of the same point with respect to the reference system resulting after the second rotation and

$[x_1^{111}\ x_2^{111}\ x_3^{111}]^T$ gives the coordinates of the same point with respect to the reference system resulting after the third rotation.

Eulerian matrix of rotation is defined as

$$\lambda = \begin{bmatrix} \lambda_{11} & \lambda_{12} & \lambda_{13} \\ \lambda_{21} & \lambda_{22} & \lambda_{23} \\ \lambda_{31} & \lambda_{32} & \lambda_{33} \end{bmatrix} \quad \textbf{(4.33)}$$

where λ_{ij}'s (i=1, 2, 3 and j=1, 2, 3) are called Eulerian angles or direction cosines and are defined as

$$\lambda_{11} = \cos(x_1^1, x_1) \quad \textbf{(4.34)}$$

$$\lambda_{12} = \cos(x_1^1, x_2) \quad \textbf{(4.35)}$$

$$\lambda_{13} = \cos(x_1^1, x_3) \quad \textbf{(4.36)}$$

$$\lambda_{21} = \cos(x_2^1, x_1) \quad \textbf{(4.37)}$$

$$\lambda_{22} = \cos(x_2^1, x_2) \quad \textbf{(4.38)}$$

$$\lambda_{23} = \cos(x_2^1, x_3) \quad \textbf{(4.39)}$$

$$\lambda_{31} = \cos(x_3^1, x_1) \quad \textbf{(4.40)}$$

$$\lambda_{32} = \cos(x_3^1, x_2) \quad \textbf{(4.41)}$$

$$\lambda_{33} = \cos(x_3^1, x_3) \quad \textbf{(4.42)}$$

It must be kept in mind that x_i's (i=1, 2, 3) are the axes of the coordinate system prior to the rotations and x_i^1's (i=1, 2, 3) are the axes of the coordinate system following the rotations.

Consider a coordinate system (x_1^1, x_2^1, x_3^1) initially coinciding with the coordinate system (x_1, x_2, x_3) and undergoing three consecutive rotations. The first rotation will take place about the first axis of the fixed reference frame (x_1), the second rotation will take place about the second axis of the reference frame resulting after the first rotation, (x_2^1), and the third rotation will take place about the third axis of the reference frame resulting after the second rotation, (x_3^{11}). If the first rotation is γ degrees, the second rotation is β degrees and the third rotation is α degrees, Eulerian angles of rotation can be derived as the following.

Since the first rotation takes place about the first axis of the fixed reference frame, x_1 is the axis of rotation, $x_1 = x_1^1$, and the elements of the first rotation matrix λ_1 are

$$\lambda_{11} = \cos(x_1^1, x_1) = \cos(x_1^1, x_1^1) = \cos 0 = 1 \quad \textbf{(4.43)}$$

$$\lambda_{12} = \cos(x_1^1, x_2) = \cos(x_1, x_2) = \cos 90 = 0 \quad \textbf{(4.44)}$$

$$\lambda_{13} = \cos(x_1^1, x_3) = \cos(x_1, x_3) = \cos 90 = 0 \quad \textbf{(4.45)}$$

$$\lambda_{21} = \cos(x_2^1, x_1) = \cos(x_2^1, x_1^1) = \cos 90 = 0 \quad \textbf{(4.46)}$$

$$\lambda_{22} = \cos(x_2^1, x_2) = \cos\gamma \quad \textbf{(4.47)}$$

$$\lambda_{23} = \cos(x_2^1, x_3) = \cos(90-\gamma) \quad \textbf{(4.48)}$$

$$\lambda_{31} = \cos(x_3^1, x_1) = \cos(x_3^1, x_1^1) = \cos 90 = 0 \quad \textbf{(4.49)}$$

$$\lambda_{32} = \cos(x_3^1, x_2) = \cos(90+\gamma) \quad \textbf{(4.50)}$$

$$\lambda_{33} = \cos(x_3^1, x_3) = \cos\gamma \quad \textbf{(4.51)}$$

Since the second rotation takes place about the second axis of the reference frame resulting after the first rotation, x_2^1 is the axis of rotation, $x_2^1 = x_2^{11}$, and the elements of the second rotation matrix λ_2 are

$$\lambda_{11} = \cos(x_1^{11}, x_1^1) = \cos(x_1^{11}, x_1) = \cos\beta \quad \textbf{(4.52)}$$

$$\lambda_{12} = \cos(x_1^{11}, x_2^1) = \cos(x_1^{11}, x_2^{11}) = \cos 90 = 0 \quad \textbf{(4.53)}$$

$$\lambda_{13} = \cos(x_1^{11}, x_3^1) = \cos(90+\beta) \quad \textbf{(4.54)}$$

$$\lambda_{21} = \cos(x_2^{11}, x_1^1) = \cos(x_2^1, x_1^1) = \cos 90 = 0 \quad \textbf{(4.55)}$$

$$\lambda_{22} = \cos(x_2^{11}, x_2^1) = \cos(x_2^1, x_2^1) = \cos 0 = 1 \quad \textbf{(4.56)}$$

$$\lambda_{23} = \cos(x_2^{11}, x_3^1) = \cos(x_2^1, x_3^1) = \cos 90 = 0 \quad \textbf{(4.57)}$$

$$\lambda_{31} = \cos(x_3^{11}, x_1^1) = \cos(90-\beta) \quad \textbf{(4.58)}$$

$$\lambda_{32} = \cos(x_3^{11}, x_2^1) = \cos(x_3^{11}, x_2^{11}) = \cos 90 = 0 \quad \textbf{(4.59)}$$

$$\lambda_{33} = \cos(x_3^{11}, x_3^1) = \cos\beta \quad \textbf{(4.60)}$$

Since the third rotation takes place about the third axis of the reference frame resulting after the second rotation, x_3^{11} is the axis of rotation, $x_3^{11} = x_3^{111}$, and the elements of the third rotation matrix λ_3 are

$$\lambda_{11} = \cos(x_1^{111}, x_1^{11}) = \cos\alpha \quad \textbf{(4.61)}$$

$$\lambda_{12} = \cos(x_1^{111}, x_2^{11}) = \cos(90-\alpha) \quad \textbf{(4.62)}$$

$$\lambda_{13} = \cos(x_1^{111}, x_3^{11}) = \cos(x_1^{111}, x_3^{111}) = \cos 90 = 0 \quad \textbf{(4.63)}$$

$$\lambda_{21} = \cos(x_2^{111}, x_1^{11}) = \cos(90+\alpha) \quad \textbf{(4.64)}$$

$$\lambda_{22} = \cos(x_2^{111}, x_2^{11}) = \cos\alpha \quad \textbf{(4.65)}$$

$$\lambda_{23} = \cos(x_2^{111}, x_3^{11}) = \cos(x_2^{111}, x_3^{111}) = \cos 90 = 0 \quad \textbf{(4.66)}$$

$$\lambda_{31} = \cos(x_3^{111}, x_1^{11}) = \cos(x_3^{11}, x_1^{11}) = \cos 90 = 0 \quad \textbf{(4.67)}$$

$$\lambda_{32} = \cos(x_3^{111}, x_2^{11}) = \cos(x_3^{11}, x_2^{11}) = \cos 90 = 0 \quad \textbf{(4.68)}$$

$$\lambda_{33} = \cos(x_3^{111}, x_3^{11}) = \cos(x_3^{111}, x_3^{111}) = \cos 0 = 1 \quad \textbf{(4.69)}$$

Trigonometric equalities state

$$\cos(a+b) = \cos a \cos b - \sin a \sin b \quad \textbf{(4.70)}$$

$$\cos(a-b) = \cos a \cos b + \sin a \sin b \quad \textbf{(4.71)}$$

Thus,

$$\cos(90-a) = \cos 90 \cos a + \sin 90 \sin a = \sin a \quad \textbf{(4.72)}$$

$$\cos(90+a) = \cos 90 \cos a - \sin 90 \sin a = -\sin a \quad \textbf{(4.73)}$$

and the three rotation matrices become

$$\lambda_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & \sin\gamma \\ 0 & -\sin\gamma & \cos\gamma \end{bmatrix} \quad \textbf{(4.74)}$$

$$\lambda_2 = \begin{bmatrix} \cos\beta & 0 & -\sin\beta \\ 0 & 1 & 0 \\ \sin\beta & 0 & \cos\beta \end{bmatrix} \quad \textbf{(4.75)}$$

$$\lambda_3 = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \textbf{(4.76)}$$

This means that if a coordinate system undergoes three such consecutive rotations, a point in the resulting frame can be given as

$$[x_1''' \; x_2''' \; x_3'''] = \lambda_3 \; \lambda_2 \; \lambda_1 \; [x_1 \; x_2 \; x_3]^T \quad (4.77)$$

where $[x_1 \; x_2 \; x_3]^T$ defines the same point in coordinates of the frame which is fixed in space, and about which the first rotation took place. In this case, $\lambda_1 \; [x_1 \; x_2 \; x_3]^T$ defines that same point in coordinates of the frame resulting after the first rotation, $\lambda_2 \; \lambda_1 \; [x_1 \; x_2 \; x_3]^T$ defines that same point in coordinates of the frame resulting after the second rotation, and finally $\lambda_3 \; \lambda_2 \; \lambda_1 \; [x_1 \; x_2 \; x_3]^T$ defines that same point in coordinates of the frame resulting after the third rotation.

In short,

$$[x_1''' \; x_2''' \; x_3'''] = \lambda_{Resultant} \; [x_1 \; x_2 \; x_3]^T \quad (4.78)$$

where

$$\lambda_{Resultant} = \lambda_3 \; \lambda_2 \; \lambda_1 \quad (4.79)$$

λ_1 is the rotation matrix of the first rotation, λ_2 is the rotation matrix of the second rotation and λ_3 is the rotation matrix of the third rotation. $\lambda_{Resultant}$ is the matrix that defines the rotation from the space-fixed coordinate frame to the body-fixed coordinate frame. In other words, when the coordinates of a point are known in the space-fixed reference frame, the coordinates of the same point in the body-fixed reference frame can be obtained effortlessly. The resultant rotation matrix is obtained by multiplying the rotation matrices in the reverse order of the rotations.

If a point is given in coordinates of the moving (body-fixed) coordinate frame, and it is desired to express it in coordinates of the space-fixed coordinate frame, which is the case in this application, inverses of the Eulerian matrices derived above must be used. In this particular application, angle bisector vectors are expressed with respect to the reference system fixed to the top platform, which might be considered as a body-fixed reference frame, and it is desired to express them with respect to the reference system fixed to the base platform, which might be considered as a space-fixed reference frame. Mathematically, if $[x_1''' \; x_2''' \; x_3''']^T$, which is the result of three consecutive rotations is known, and it is desired to obtain the coordinates of the point before the rotations, with respect to the space-fixed reference frame, $[x_1 \; x_2 \; x_3]^T$, then

$$[x_1\ x_2\ x_3]^T = \lambda_3^{-1}\ \lambda_2^{-1}\ \lambda_1^{-1}\ [x_1^{111}\ x_2^{111}\ x_3^{111}]^T \quad \textbf{(4.80)}$$

where λ_3^{-1} is the inverse of the third rotation matrix,

λ_2^{-1} is the inverse of the second rotation matrix, and

λ_1^{-1} is the inverse of the first rotation matrix.

Thus, the matrix that defines the rotation from the reference frame fixed to the top platform of the SPM to the reference frame fixed to the base platform will be obtained by multiplying the inverses of the rotation matrices in the reverse order of the rotations. In this application, the rotation matrix needed to convert angle bisector vectors from coordinates of the reference frame fixed to the top platform into the coordinates of the base platform can be expressed as

$$R = R_z\ R_y\ R_x \quad \textbf{(4.81)}$$

where R_x is the inverse of the Eulerian matrix defining the rotation about the x axis,

R_y is the inverse of the Eulerian matrix defining the rotation about the y axis, and

R_z is the inverse of the Eulerian matrix defining the rotation about the z axis.

Mathematically,

$$R_x = \lambda_1^{-1} \quad \textbf{(4.82)}$$

$$R_y = \lambda_2^{-1} \quad \textbf{(4.83)}$$

$$R_z = \lambda_3^{-1} \quad \textbf{(4.84)}$$

One other point to note is that the rotation matrices defined above are orthogonal matrices, meaning their transposes are equal to their inverses such as

$$\lambda_1^{-1} = \lambda_1^T \quad \textbf{(4.85)}$$

$$\lambda_2^{-1} = \lambda_2^T \quad \textbf{(4.86)}$$

$$\lambda_3^{-1} = \lambda_3^T \quad \textbf{(4.87)}$$

and thus

$$\lambda_1\ \lambda_1^T = I \quad \textbf{(4.88)}$$

$$\lambda_2 \ \lambda_2^T = I \quad (4.89)$$

$$\lambda_3 \ \lambda_3^T = I \quad (4.90)$$

where I is the identity matrix.

4.4 Loop Method

This method fine-tunes the result of the FK solution and further decreases the error function defined above. In this method, after the training is complete, an input vector L in the form $[l_1, l_2, l_3, l_4, l_5, l_6]$ is fed to the NN to calculate an output vector in the form $[x, y, z, \gamma, \beta, \alpha]$, which then is sent to the IK equations to calculate a new input vector L_i. The result of the IK equations are compared with the initial input vector L. The offset between the actual input vector L and the vector L_i produced by the IK equations is found as ΔL. The initial input vector is adjusted as

$$L_{new} = L - \Delta L / n\_loop \quad (4.91)$$

where n_loop is the number of loops to complete. One loop is defined as taking one set of leg lengths, propagating it through the NN to produce an output configuration of the SPM, sending the result to the IK equations, calculating the offset between the actual input lengths and the leg lengths calculated by the IK equations, adjusting the leg lengths and sending the adjusted leg lenghts once more to the NN to produce the final position and orientation of the top platform. It was found out that increasing the number of loops increases the performance of the method. Computational time of the loop method is in the order of ms, thus it can be used in real-time applications. Programming has been done in the Fortran language. Figure 4.3 illustrates how the loop method works.

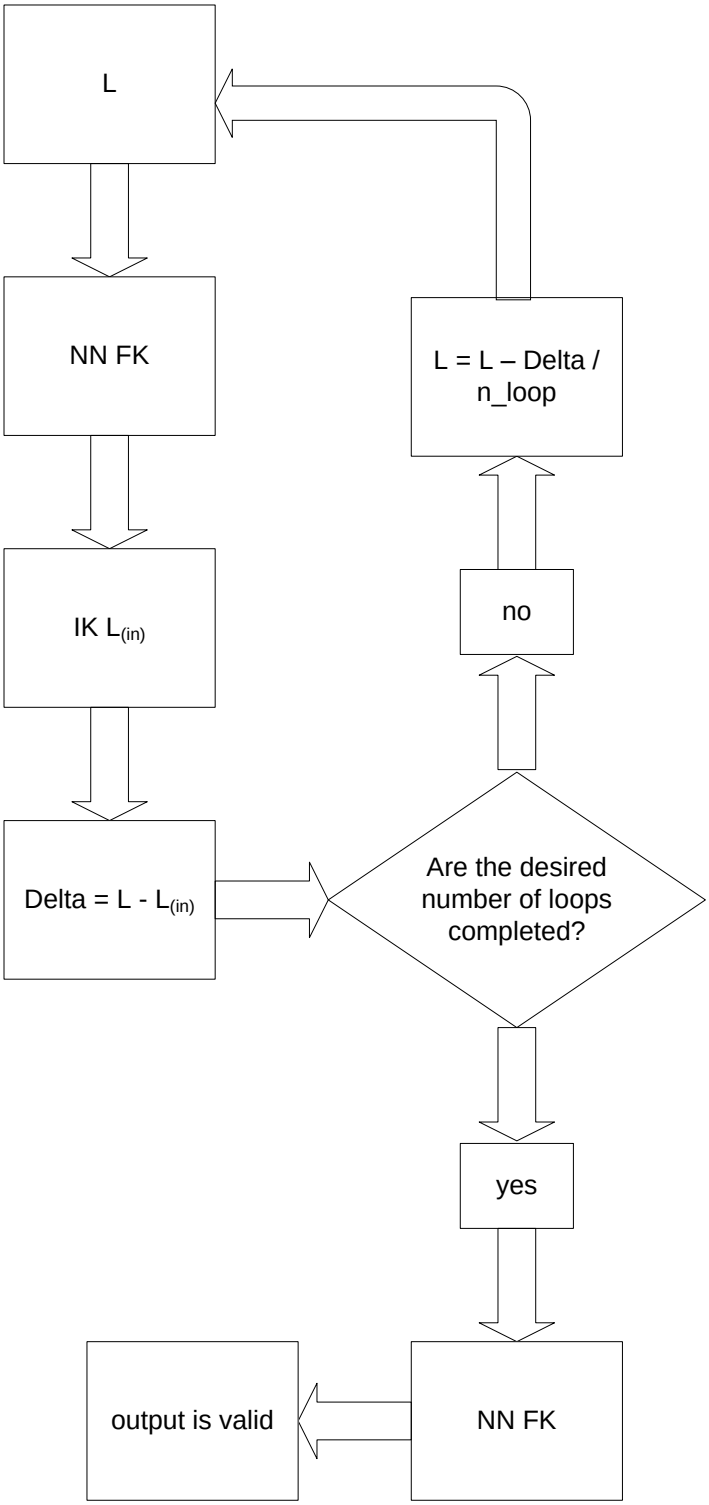


Figure 4.3: The Loop Method

5. RESULTS AND DISCUSSIONS

5.1 Purely Rotational Case

Training of the purely rotational SPM by using 17 123 data patterns obtained by fixing the center of gravity to [0 0 0.8] (meters) and rotating the top platform about the x, y, and z axes within the ranges $\pm 20°$, $\pm 22°$, and $\pm 24°$, respectively, with stepsizes of $1°$ was performed until the average error per data pattern was reduced to 4.91×10^{-3} radians, meaning $0.28°$. This can be considered a very accurate result for most applications. The next step was to further decrease the average error per data set by applying the loop method described in section 4.4. It was seen that applying the loop method even only once decreased the error per data pattern to 2.13×10^{-4} radians, meaning the order of magnitude of the error drops after the application of the loop method. Table 5.1 and Figure 5.1 show the effect of applying the loop method on the average error per data pattern.

Table 5.1: Effect of the number of loops on the performance of the NN when testing with data with stepsizes of $1°$

number of loops	error per data pattern (radians)	error per data pattern
0	4.91×10^{-3}	0.2814650
1	2.13×10^{-4}	0.0122102
2	1.63×10^{-4}	0.0093439
3	1.46×10^{-4}	0.0083694
4	1.38×10^{-4}	0.0079108
5	1.33×10^{-4}	0.0076242
10	1.23×10^{-4}	0.0070510
15	1.20×10^{-4}	0.0068790
20	1.18×10^{-4}	0.0067643
30	1.16×10^{-4}	0.0066497
40	1.16×10^{-4}	0.0066497

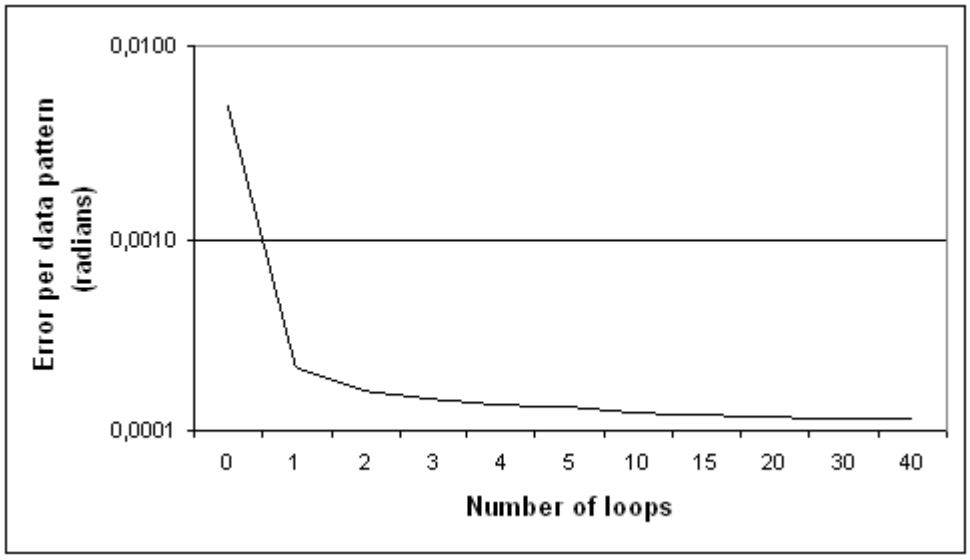


Figure 5.1: Effect of the loop method on the error per data pattern when testing with data with stepsizes of 1°

Next step was to test the NN by using 136 893 data patterns that were formed by scanning the mentioned workspace with stepsizes of 0.5°. It is seen by inspecting Table 5.2 and Figure 5.2 that the loop method drops the average error per data pattern drastically. Average error per data set, before the application of the loop method was 4.91×10^{-3} radians, and it dropped to 2.13×10^{-4} radians after applying 1 loop. Table 5.2 and Figure 5.2 show the effect of the loop method on the performance of the NN when testing with data with stepsizes of 0.5°.

Table 5.2: Effect of the number of loops on the performance of the NN when testing with data with stepsizes of 0.5°

number of loops	error per data pattern (radians)	error per data pattern
0	4.91×10^{-3}	0.2814650
1	2.13×10^{-4}	0.0122102
2	1.63×10^{-4}	0.0093439
3	1.46×10^{-4}	0.0083694
4	1.38×10^{-4}	0.0079108
5	1.33×10^{-4}	0.0076242
10	1.23×10^{-4}	0.0070510
15	1.20×10^{-4}	0.0068790
20	1.18×10^{-4}	0.0067643
30	1.16×10^{-4}	0.0066497
40	1.15×10^{-4}	0.0065924

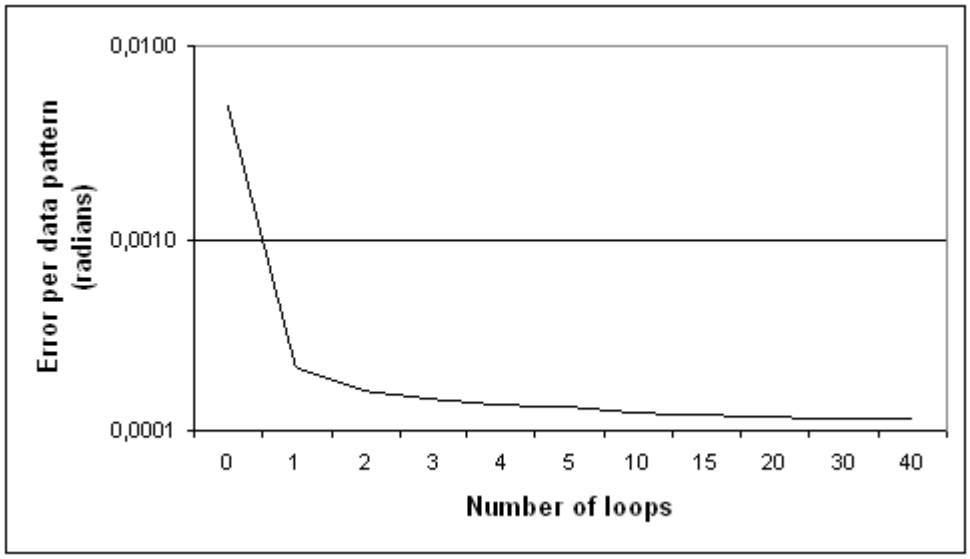


Figure 5.2: Effect of the loop method on the error per data pattern when testing with data with stepsizes of 0.5°

Last step was to test the NN by using 1 093 886 data patterns that were formed by scanning the mentioned workspace with stepsizes of 0.25°. It is seen by inspecting Table 5.3 and Figure 5.3 that the loop method drops the average error per data pattern drastically. Average error per data set, before the application of the loop method was 4.91×10^{-3} radians, and it dropped to 2.13×10^{-4} radians after applying 1 loop. Table 5.3 and Figure 5.3 show the effect of the loop method on the performance of the NN when testing with data with stepsizes of 0.25°.

Table 5.3: Effect of the number of loops on the performance of the NN when testing with data with stepsizes of 0.25°

number of loops	error per data pattern (radians)	error per data pattern
0	4.91×10^{-3}	0.2814650
1	2.13×10^{-4}	0.0122102
2	1.63×10^{-4}	0.0093439
3	1.46×10^{-4}	0.0083694
4	1.38×10^{-4}	0.0079108
5	1.33×10^{-4}	0.0076242
10	1.23×10^{-4}	0.0070510
15	1.20×10^{-4}	0.0068790
20	1.18×10^{-4}	0.0067643
30	1.16×10^{-4}	0.0066497
40	1.16×10^{-4}	0.0066497

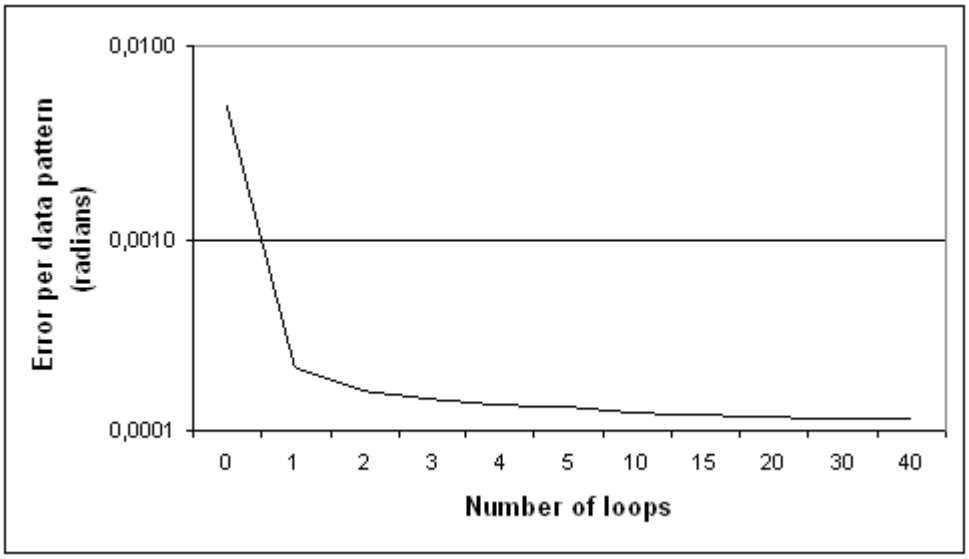


Figure 5.3: Effect of the loop method on the error per data pattern when testing with data with stepsizes of 0.25 °

Tables 5.1-5.3 and Figures 5.1-5.3 reveal that the order of magnitude of the average error per data pattern drops 10^{-4} radians after the application of the loop method, which is only 0.005 °. This is considered very accurate in many applications. It is shown by training the NN with data patterns obtained by scanning the workspace with stepsizes of 1 ° and testing the NN with data patterns obtained by scanning the workspace with stepsizes of 0.5 ° and 0.25 ° that the NN is capable of producing accurate outputs for input patterns that were not in the training set. Weights and biases obtained after training with purely rotational data are given in Appendix A. Results of testing with one purely rotational data pattern up to 9 loops are given in Appendix B.

5.2 Purely Translational Case

Training of the purely translational SPM by using 64 803 data patterns obtained by moving the top platform of the SPM in the x, y, and z directions within the ranges ± 450 mm, ± 390 mm and (600 mm, 930 mm) with stepsizes of 10 mm was performed until the average error per data pattern was reduced to 8.62×10^{-3} m, meaning 8.62 mm. The next step was to further decrease the average error per data set by applying the loop method described in section 4.4. It was seen that applying the loop method even only once decreased the error per data set to 3.14×10^{-4} m, meaning the order of magnitude of the error drops after the application of the loop method. Table 5.4 and

Figure 5.4 show the effect of applying the loop method on the average error per data pattern.

Table 5.4: Effect of the number of loops on the performance of the NN when testing with data with stepsizes of 10 mm

number of loops	error per data pattern (meters)
0	8.62×10^{-3}
1	3.14×10^{-4}
2	2.37×10^{-4}
3	2.11×10^{-4}
4	1.98×10^{-4}
5	1.90×10^{-4}
10	1.75×10^{-4}
15	1.70×10^{-4}
20	1.67×10^{-4}
30	1.65×10^{-4}
40	1.63×10^{-4}

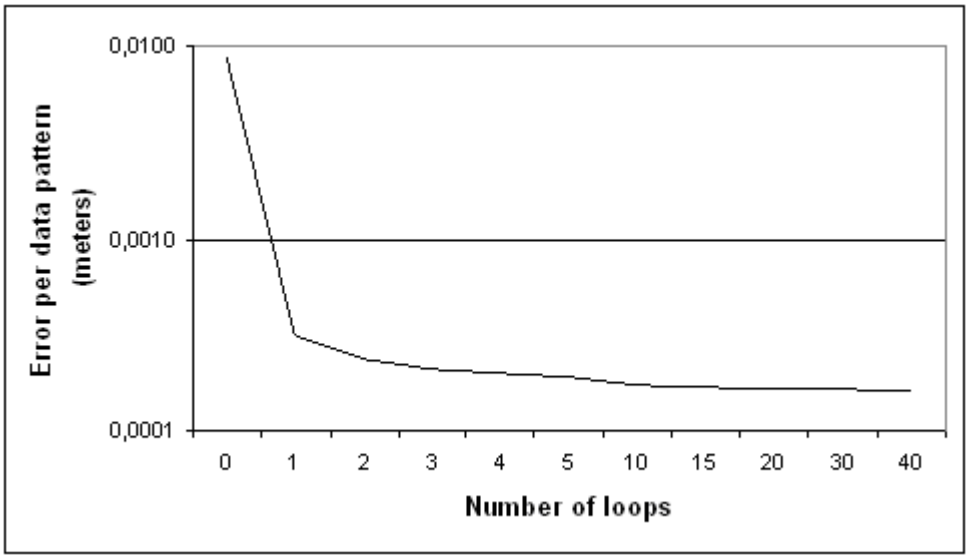


Figure 5.4: Effect of the loop method on the error per data pattern when testing with data with stepsizes of 10 mm

Next step was to test the NN by using 516 811 data patterns that were formed by scanning the mentioned workspace with stepsizes of 5 mm. It is seen by inspecting Table 5.5 and Figure 5.5 that the loop method drops the average error per data pattern drastically. Average error per data set, before the application of the loop method was 8.61×10^{-3} meters, and it dropped to 3.14×10^{-4} meters after applying 1 loop. Table 5.5 and Figure 5.5 show the effect of the loop method on the performance of the NN when testing with data with stepsizes of 5 mm.

Table 5.5: Effect of the number of loops on the performance of the NN when testing with data with stepsizes of 5 mm

number of loops	error per data pattern (meters)
0	8.61×10^{-3}
1	3.14×10^{-4}
2	2.37×10^{-4}
3	2.12×10^{-4}
4	1.99×10^{-4}
5	1.90×10^{-4}
10	1.75×10^{-4}
15	1.70×10^{-4}
20	1.68×10^{-4}
30	1.65×10^{-4}
40	1.64×10^{-4}

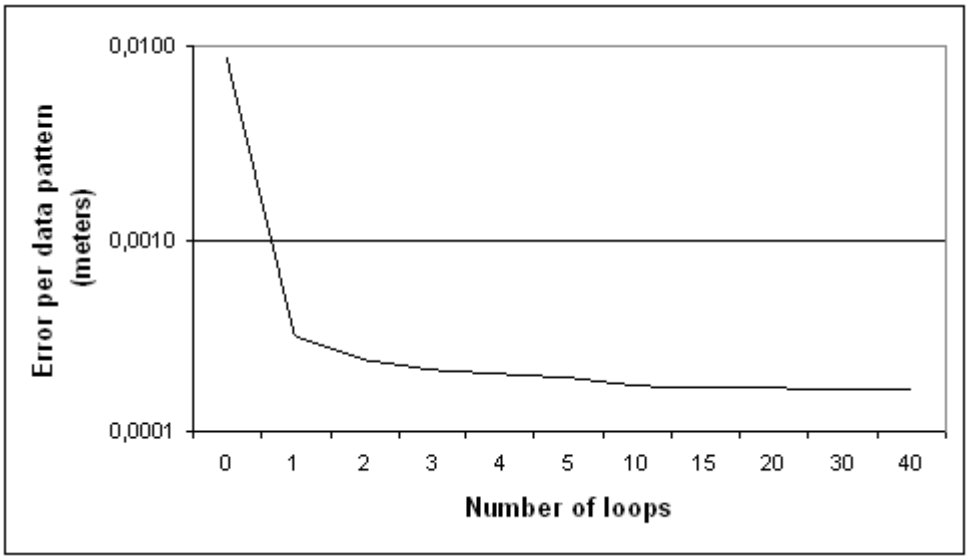


Figure 5.5: Effect of the loop method on the error per data pattern when testing with data with stepsizes of 5 mm

Last step was to test the NN by using 4 130 586 data patterns that were formed by scanning the mentioned workspace with stepsizes of 2.5 mm. It is seen by inspecting Table 5.6 and Figure 5.6 that the loop method drops the average error per data pattern drastically. Average error per data set, before the application of the loop method was 8.66×10^{-3} meters, and it dropped to 3.15×10^{-4} meters after applying 1 loop. Table 5.6 and Figure 5.6 show the effect of the loop method on the performance of the NN when testing with data with stepsizes of 2.5 mm.

Table 5.6: Effect of the number of loops on the performance of the NN when testing with data with stepsizes of 2.5 mm

number of loops	error per data pattern (meters)
0	8.66×10^{-3}
1	3.15×10^{-4}
2	2.38×10^{-4}
3	2.12×10^{-4}
4	1.99×10^{-4}
5	1.86×10^{-4}
10	1.75×10^{-4}
15	1.71×10^{-4}
20	1.68×10^{-4}
30	1.65×10^{-4}
40	1.64×10^{-4}

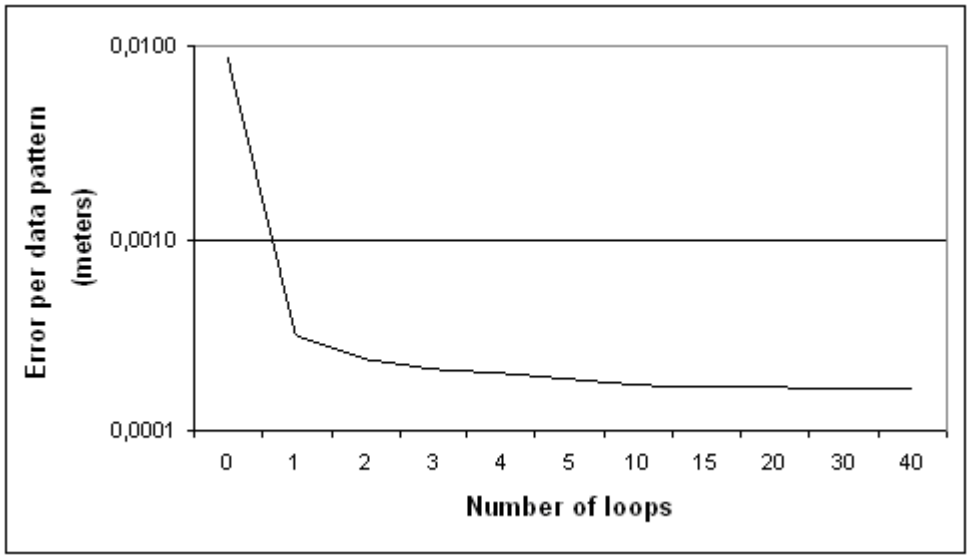


Figure 5.6: Effect of the loop method on the error per data pattern when testing with data with stepsizes of 2.5 mm

Tables 5.4-5.6 and Figures 5.4-5.6 reveal that the order of magnitude of the average error per data pattern drops to 10^{-4} m after the application of the loop method, which is only one tenth of a mm. This is considered very accurate in many applications. NN is capable of producing accurate outputs for input patterns that were not in the training set. Weights and biases obtained after training with purely translational data are given in Appendix C. Results of testing with one purely translational data pattern up to 9 loops are given in Appendix D.

5.3 General Case

Training of the general SPM by using 53 415 data patterns obtained by moving the top platform of the SPM randomly in the x, y, and z directions and rotating it about the x, y, and z axes within the ranges ± 200 mm in the x direction, ± 200 mm in the y direction, (750 mm, 950 mm) in the z direction, and ± 10 ° about the x, y, and z axes was performed until the average error per data pattern was reduced to 1.972×10^{-2}. It should be noted that this error does not have a valid unit since it is the combination of translational and rotational errors. The next step was to further decrease the average error per data set by applying the loop method described in section 4.4. It was seen that applying the loop method even only once decreased the error per data set to 4.800×10^{-3}, meaning the order of magnitude of the error drops after the application of the loop method. Table 5.7 and Figure 5.7 show the effect of applying the loop method on the average error per data pattern.

Table 5.7: Effect of the number of loops on the performance of the NN

number of loops	error per data pattern
0	1.972 x 10^{-2}
1	4.800 x 10^{-3}
2	3.922 x 10^{-3}
3	3.638 x 10^{-3}
4	3.494 x 10^{-3}
5	3.407 x 10^{-3}
10	3.229 x 10^{-3}
15	3.169 x 10^{-3}
20	3.138 x 10^{-3}
30	3.108 x 10^{-3}
40	3.093 x 10^{-3}

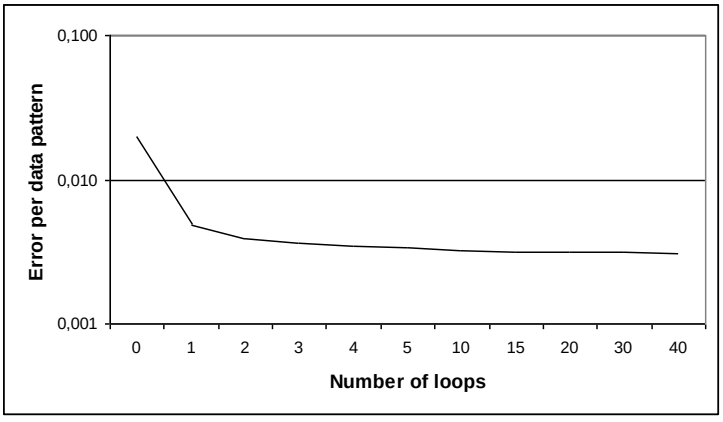


Figure 5.7: Effect of the loop method on the error per data pattern

Next step was to test the NN by using 106 443 data patterns that were randomly formed by moving the top platform of the SPM randomly in the x, y, and z directions and rotating it about the x, y, and z axes within the ranges ± 200 mm in the x direction, ± 200 mm in the y direction, (750 mm, 950 mm) in the z direction, and ± 10 ° about the x, y, and z axes, as in the training data. It is seen by inspecting Table 5.8 and Figure 5.8 that the loop method drops the average error per data pattern drastically. Average error per data set, before the application of the loop method was 1.968×10^{-2}, and it dropped to 4.798×10^{-3} after applying 1 loop. Table 5.8 and Figure 5.8 show the effect of the loop method on the performance of the NN when testing with randomly generated data patterns that were not in the training set.

Table 5.8: Effect of the number of loops on the performance of the NN when testing with randomly generated data

number of loops	error per data pattern
0	1.968×10^{-2}
1	4.798×10^{-3}
2	3.919×10^{-3}
3	3.663×10^{-3}
4	3.490×10^{-3}
5	3.403×10^{-3}
10	3.225×10^{-3}
15	3.165×10^{-3}
20	3.135×10^{-3}
30	3.104×10^{-3}
40	3.089×10^{-3}

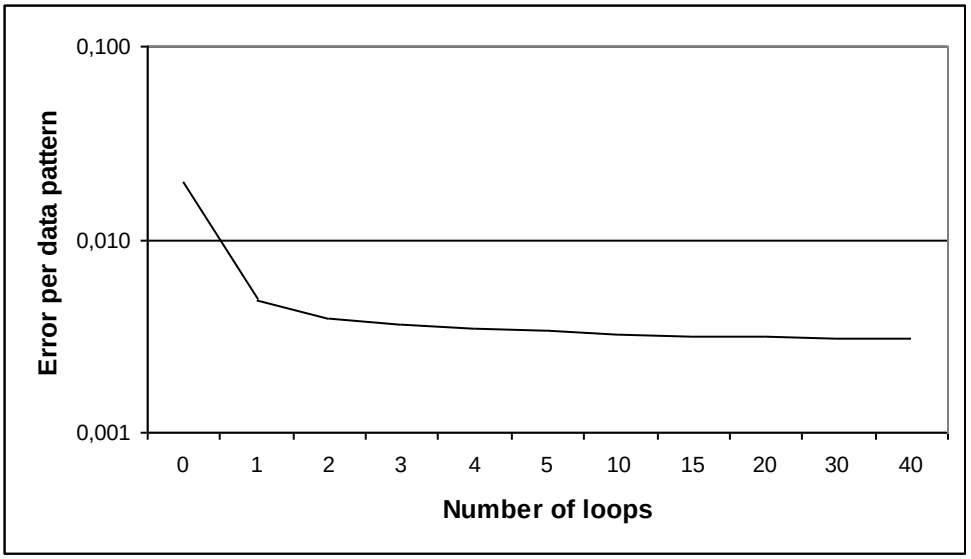


Figure 5.8: Effect of the loop method on the error per data pattern when testing with randomly generated data

Tables 5.7 and 5.8 and Figures 5.7 and 5.8 reveal that the order of magnitude of the average error per data pattern drops to 10^{-3} after the application of the loop method. This is considered very accurate in many applications. NN is capable of producing accurate outputs for input patterns that were not in the training set. Weights and biases obtained after training with general spatial data are given in Appendix E. Results of testing with one general data pattern up to 9 loops are given in Appendix F.

6. CONCLUSION

A neural network has been applied to the forward kinematics problem of the purely rotational, purely translational, and the 6 degree of freedom general motions of the 6-3 Stewart platform mechanism. For the rotational case, the neural network has been trained by using data patterns formed by fixing the center of gravity of the top platform to [0 0 0.8] (meters) and scanning the rotational workspace with stepsizes of 1° about the x, y, and z axes and tested by using data patterns formed by scanning the workspace with stepsizes of 0.5° and 0.25° about x, y, and z directions. For the translational case, the neural network has been trained by using data patterns formed by scanning the workspace with stepsizes of 10 mm in x, y, and z directions while keeping the top platform parallel to the base and tested by using data patterns formed by scanning the workspace with stepsizes of 5 mm and 2.5 mm in x, y, and z directions. For the general case, the neural network has been trained and tested by using randomly generated data patterns. In all cases, data sets have been obtained by moving the top platform of the mechanism in a given workspace, solving the inverse kinematics equations and then checking to see whether the leg lengths are within the allowed limits. After the training of the network was complete, average errors per data pattern were reduced to 10^{-3} radians, 10^{-3} m, and 10^{-2} in the purely rotational, purely translational and, general cases, respectively. It was seen that using inverse kinematics equations is helpful in fine-tuning the solution and decreasing the average error per data pattern to the order of 10^{-4} radians, 10^{-4} m, and 10^{-3} in the purely rotational, purely translational cases, and general cases, respectively, which can be considered very accurate in many applications. Results of this study are that a neural network might be trained to estimate the position and orientation of the top platform of the SPM within an allowed error margin, instead of solving a polynomial of order 16 and applying mechanical and geometrical tests to find the real physical solution. It is easier to train an artificial neural network for a particular architecture and then simply finding the right outputs, namely the position of the center of gravity of the top platform with respect to the coordinate system at the base and orientation of the

top platform with respect to the base platform, for a set of given inputs, namely leglengths. Once the network is trained, the method yields accurate results fast enough to solve the forward kinematics of the Stewart platform mechanism so that it is applicable for real-time situations and can be used in real systems instead of a position sensor and a gyroscope. The loop method is a simple procedure and helps in obtaining more accurate results by using the inverse kinematics equations. Thus, this thesis has shown that an artificial neural network might be used in solving the forward kinematics of a Stewart Platform Mechanism, instead of dealing with a polynomial of order 16, and the loop method might be used to obtain more accurate results. Following from this work, possibilities for future works include, but are not limited to, using epoch based training so that training takes a shorter amount of time, using genetic algorithms instead of using backpropagation of errors so that the weights and biases always converge to values which will give the global minimum of the error, parallel programming so that more data patterns might be used in training, and using experimental data sets in training and testing the network so that more realistic results are obtained.

REFERENCES

[1] **Stewart, D,** 1965. A Platform with Six Degrees of Freedom, *Proc. Instn. Mech. Engrs*, **180**(15), 371-386.

[2] **Ku, D,** 1999. Direct Displacement Analysis of a Stewart Platform Mechanism, *Mechanism and Machine Theory*, **34**(3), 453-465.

[3] **Merlet, J. P.,** 1993. Direct Kinematics of Parallel Manipulators, *IEEE Transactions on Robotics and Automation*, **9**(6), 842-846.

[4] **Yiu, Y. K, Cheng, H, Xiong, Z. H, Liu G. F., and Li, Z. X**, 2001. On the Dynamics of Parallel Manipulators, *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, Seoul, Korea, May 21-26, 3766-3771.

[5] **Liu, K, Fitzgerald J. M, and Lewis, F. L,** 1993. Kinematic Analysis of a Stewart Platform Manipulator, *IEEE Transactions on Industrial Electronics*, **40**(2), 282-293.

[6] **Lee, K. and Shah, D. K.** 1988. Kinematic Analysis of a Three-Degrees-of-Freedom In-Parallel Actuated Manipulator, *IEEE Journal of Robotics and Automation*, **4**(3), 354-360.

[7] **Shi, X. and Fenton, R. G,** 1992. Solution to the Forward Instantaneous Kinematics for a General 6-dof Stewart Platform, *Mechanism and Machine Theory*, **27**(3), 251-259.

[8] **Innocenti, C, and Parenti-Castelli, V.**, 1990. Direct Position Analysis of the Stewart Platform Mechanism, *Mechanism and Machine Theory*, **25**(6), 611-621.

[9] **Dasgupta, B, and Mruthyunjaya, T. S.,** 2000. The Stewart Platform Manipulator: A Review, *Mechanism and Machine Theory*, **35**, 15-40.

[10] **Romdhane, L.,** 1999. Design and Analysis of a Hybrid Serial-Parallel Manipulator, *Mechanism and Machine Theory*, **34**, 1037-1055.

[11] **Arai, T., Yuasa, K., Mae, Y., Inoue, K, Miyawaki, K, and Koyachi, N**, 2002. A Hybrid Drive Parallel Arm for Heavy Material Handling, *IEEE Robotics and Automation Magazine*, 45-54.

[12] **Stoughton, R. S., and Arai, T.,** 1993. A Modified Stewart Platform Manipulator with Improved Dexterity, *IEEE Transactions on Robotics and Automation*, **9**(2), 166-173.

[13] **Nanua, P., Waldron, K. J., and Murthy, V.**, 1990. Direct Kinematic Solution of a Stewart Platform, *IEEE Transactions on Robotics and Automation*, **6**(4), 438-444.

[14] **Wohlhart, K.,** 1994. Displacement Analysis of the General Spherical Stewart Platform, *Mechanism and Machine Theory*, **29**(4), 581-589.

[15] **Dasgupta, B, and Mruthyunjaya, T. S.**, 1994. A Canonical Formulation of the Direct Position Kinematics Problem for a General 6-6 Stewart Platform, *Mechanism and Machine Theory*, **29**(6), 819-827.

[16] **Dasgupta, B, and Mruthyunjaya, T. S.**, 1996. A Constructive Predictor-Corrector Algorithm for the Direct Position Kinematics Problem for a General 6-6 Stewart Platform, *Mechanism and Machine Theory*, **31**(6), 799-811.

[17] **Zhao, X, and Peng, S.,** 2000. Direct Displacement Analysis of Parallel Manipulators, *Journal of Robotic Systems*, **17**(6), 341-345.

[18] **Jakobovic, D, and Jelenkovic, L.,** 2002. The Forward and Inverse Kinematics Problems for Stewart Parallel Mechanisms, *International Scientific Conference on Production Engineering*, Croatia, **II**, 1-12.

[19] **Tsai, M. S., Shiau, T. N, Tsai Y. J., and Chang, T. H,** 2003. Direct Kinematic Analysis of a 3-PRS Parallel Mechanism, *Mechanism and Machine Theory*, **38**, 71-83.

[20] **Kim, J., and Park, F. C.,** 2001. Direct Kinematic Analysis of 3-RS Parallel Mechanisms, *Mechanism and Machine Theory*, **36**, 1121-1134.

[21] **Di Gregorio, R,** 2001. Kinematics of a New Spherical Parallel Manipulator with Three Equal Legs: The 3-URC Wrist, *Journal of Robotic Systems*, **18**(5), 213-219.

[22] **Callegari, M, and Tarantini, M,** 2003. Kinematic Analysis of a Novel Translational Platform, *Transactions of the ASME*, **125**, 308-315.

[23] **Di Gregorio, R**, 2002. Translational Parallel Manipulators: New Proposals, *Journal of Robotic Systems*, **19**(12), 595-603.

[24] **Dasgupta, B and Mruthyunjaya T. S.**, 1998. Closed-Form Dynamic Equations of the General Stewart Platform Through the Newton-Euler Approach, *Mechanism and Machine Theory*, **33**, 993-1012.

[25] **Dasgupta, B and Mruthyunjaya T. S.,** 1998. Newton-Euler Formulation for the Inverse Dynamics of the Stewart Platform Manipulator, *Mechanism and Machine Theory*, **33**, 1135-1152.

[26] **Wang, J., and Gosselin, C. M**, 1998. A New Approach for the Dynamic Analysis of Parallel Manipulators, *Multibody System Dynamics*, **2**, 317-334.

[27] **Lebret, G, Liu, K., and Lewis, F. L.,** 1993. Dynamic Analysis and Control of a Stewart Platform Manipulator, *Journal of Robotic Systems*, **10**(5), 629-655.

[28] **Pang, H, and Shahinpoor, M.,** 1994. Inverse Dynamics of a Parallel Manipulator, *Journal of Robotic Systems*, **11**(8), 693-702.

[29] **Liu, M J., Li, C X., Li, C N,** 2000. Dynamics Analysis of the Gough-Stewart Platform Manipulator, *IEEE Transactions on Robotics and Automation*, **16**(1).

[30] **Di Gregorio, R**, 2002. Design and Singularity Analysis of a 3-Translational-DOF In-Parallel Manipulator, *Journal of Robotic Systems*, 19(1), 37–43.

[31] **Pernkopf, F., and Husty, M.,** 2002. Workspace Analysis of Stewart-Gough Manipulators Using Orientation Plots, *Proceedings of MUSME 2002, the International Symposium on Multibody Systems and Mechatronics*, Mexico City, September 12-14, paper no: 33.

[32] **Dafaoui, El- M, Amirat, Y., Pontnau, J., and François, C.,** 1998. Singularity Analysis of Three-Legged, Six DOF Platform Manipulators with RRRS Legs, *IEEE Transactions on Robotics and Automation*, **14**(1), 32-36.

[33] **Carretero, J. A, Nahon, M, and Podhoreski, R P.,** 1998. Parallel Mechanisms with Adjustable Link Parameters, *Proceedings of the 1998 IEEE/RSI International Conference on Intelligent Robots and Systems*, Victoria, October, 671-676.

[34] **Joshi, S. A, and Tsai, L. W,** 2002. Jacobian Analysis of Limited-Dof Parallel Manipulators, *Transactions of the ASME*, **124**, 254-258.

[35] **Gosselin, C, and Angeles, J.,** 1990. Analytical Study of Stewart Platforms Workspaces, *IEEE Transactions on Robotics and Automation*, **6**(3), 281-290.

[36] **Pernkopf, F., and Husty, M L.,** 2002. Singularity Analysis of Spatial Stewart-Gough Platforms with Planar Base and Platform, *Proceedings of DETC'02 2002 ASME Design Engineering Technical Conference*, Montreal, Canada, September 29-October 2, 1-8.

[37] **Collins, C L., and McCarthy, J. M,** 1997. The Singularity Loci of Two Triangular Parallel Manipulators, *ICAR*, Monterey, July 7-9, 473-478.

[38] **Angeles, J., Yang, G., and Chen, I. M,** 2003. Singularity Analysis of Three-Legged, Six-DOF Platform Manipulators with URS Legs, *IEEE/ASME Transactions on Mechatronics*, **8**(4), 469-475.

[39] **Kim, D., and Chung, W.,** 1999. Analytic Singularity Equation and Analysis of Six-Dof Parallel Manipulators Using Local Structurization Method, *IEEE Transactions on Robotics and Automation*, **15**(4), 612-622.

[40] **Bessala, J., Bidaud, P., and Ben Ouezdou, F.,** 1996. *Proceedings of the 1996 IEEE International Conference on Robotics and Automation.*

[41] **Romdhane, L., Affi, Z., and Fayet, M.,** 2002. Design and Singularity Analysis of a 3-Translational-DOF In Parallel Manipulator, *Journal of Mechanical Design*, **124**, 419-426.

[42] **Zlatanov, D, Fenton R. G., and Benhabib, B.,** 1994. Singularity Analysis of Mechanisms and Robots via A Motion-Space Model of the Instantaneous Kinematics, *IEEE*, 986-991.

[43] **Zlatanov, D, Fenton R. G., and Benhabib, B**, 1994. Singularity Analysis of Mechanisms and Robots via A Velocity-Equation Model of the Instantaneous Kinematics, *IEEE*, 986-991.

[44] **Huang, Z., Chen L. H., and Li, Y. W.,** 2003. The Singularity Principle and Property of Stewart Parallel Manipulator, *Journal of Robotic Systems*, **20**(4), 163-176.

[45] **Du Plessis, L. J., Snyman, J. A.,** 2001. A Numerical Method for the Determination of Dexterous Workspaces of Gough-Stewart Platforms, *International Journal for Numerical Methods in Engineering*, **52**, 345-369.

[46] **Di Gregorio, and R., Parenti-Castelli, V.,** 2002. Mobility Analysis of the 3-UPU Parallel Mechanism Assembled for a Pure Translational Motion, *Journal of Mechanical Design*, **124**, 259-264.

[47] **Yang, G, Chen, I. M., Lin, W., and Angeles, J.,** 2001. Singularity Analysis of Three-Legged Parallel Robots Based on Passive-Joint Velocities, *IEEE Transactions on Robotics and Automation*, **17**(4), 413-422.

[48] **Liu, X. J., Kim, J., and Oh, K. K.,** 2003. Singularity Analysis of the HALF Manipulator with Revolute Actuators, *Proceedings of the 2003 IEEE International Conference on Robotics and Automation*, Taipei, September 14-19, 767-772.

[49] **Liu, G., Lou, Y., and Li, Z.,** 2003. Singularities of Parallel Manipulators: A Geometric Treatment, *IEEE Transactions on Robotics and Automation*, **19**(4), 579-594.

[50] **Kim, D. L., Chung, W. K., and Youm, Y.,** 1997. Geometrical Approach for the Workspace of 6-Dof parallel Manipulators, *Proceedings of the 1997 IEEE International Conference on Robotics and Automation.*

[51] **Wen, J. T., and O'Brien, J. F.,** 2003. Singularities in Three-Legged Platform-Type Parallel Mechanisms, *IEEE Transactions on Robotics and Automation*, **19**(4), 720-726.

[52] **Wolf, A., and Shoham, M.,** 2003. Investigation of Parallel Manipulators Using Linear Complex Approximation, *Transactions of the ASME*, **125**, 564-572.

[53] **Wang, W., and Min, T.**, 2001. Characterization of the Analytical Boundary of the Workspace for 3-6 SPS Parallel Manipulator, *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, Seoul, May 21-26, 3755-3759.

[54] **Wang, Z., Wang, Z., Liu, W., and Lei, Y.,** 2001. A Study on Workspace, Boundary Workspace Analysis and Workpiece Positioning for Parallel Machine Tools, *Mechanism and Machine Theory*, **36**, 605-622.

[55] **Angeles, J., Yang, G., and Chen, I. M.,** 2001. Singularity Analysis of Three-Legged, Six-DOF Platform Manipulators with RRRS Legs, *2001 IEEE/ASME International Conference on Advanced Intelligent Mechatronics Proceedings*, Como, July 8-12, 32-36.

[56] **Majid, M. Z. A., Huang, Z., and Yao, Y. L.,** 2000. Workspace Analysis of a Six-Degrees of Freedom, Three-Prismatic-Prismatic-Spheric-Revolute Parallel Manipulator, *International Journal of Advanced Manufacturing Technology*, **16**, 441-449.

[57] **Merlet, J. P.**, 1995. Determination of the Orientation Workspace of Parallel Manipulators, *Journal of Intelligent and Robotic Systems*, **13**, 143-160.

[58] **Badescu, M., and Mavroidis, C.,** 2004. Workspace Optimization of 3-Legged UPU and UPS Parallel Platforms with Joint Constraints, *Journal of Mechanical Design*, **126**, 291-300.

[59] **Badescu, M., Morman, J., and Mavroidis, C.,** 2002. Workspace Optimization of 3-UPU Parallel Platforms with Joint Constraints, *Proceedings of the 2002 IEEE International Conference on Robotics and Automation*, Washington, May, 3678-3683.

[60] **Bonev, I. A., and Ryu, J.,** 2001. A new Approach to Orientation Workspace Analysis of 6-DOF Parallel Manipulators, *Mechanism and Machine Theory*, **36**, 15-28.

[61] **Li, Y., Huang, Z., and Chen, L.,** 2003. Singular Loci Analysis of 3/6-Stewart Manipulator by Singularity-Equivalent Mechanism, *Proceedings of the 2003 IEEE International Conference on Robotics and Automation*, 1881-1886.

[62] **Merlet, J. P.,** 1996. Workspace-Oriented Methodology for Designing a Parallel Manipulator, *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, Minneapolis, April, 3726-3731.

[63] **Monsarrat, B., and Gosselin, C. M.**, 2003. Workspace Analysis and Optimal Design of a 3-Leg 6-DOF Parallel Platform Mechanism, *IEEE Transactions on Robotics and Automation*, **19**(6), 954-966.

[64] **Jin, Y., Chen, I. M., and Yang, G.**, 2004. Structure Synthesis and Singularity Analysis of a Parallel Manipulator Based on Selective Actuation, *Pr. of the 2004 IEEE International Conference on Robotics and Automation*, 4533-4538.

[65] **Lee, S. H., Song, J. B., Choi, W. C., and Hong, D.,** 2002. Workspace and Force-Moment Transmission of a Variable Arm Type Parallel Manipulator, *Proceedings of the 2002 IEEE International Conference on Robotics and Automation*, Washington, May, 3666-3671.

[66] **Chen, I. M., Angeles, J., Theingi, and Oh, C. L.,** 2003. The Management of Parallel-Manipulator Singularities Using Joint Coupling, *Proceedings of the 2003 IEEE International Conference on Robotics and Automation*, Taipei, September 14-19, 773-778.

[67] **Arai, T., Takayama, K., Inoue, K., Mae, Y., and Koseki, Y.,** 2000. Parallel Mechanisms with Adjustable Link Parameters, *Proceedings of the 2000 IEEE/RSJ, International Conference on Intelligent Robots and Systems*, 671-676.

[68] **Miller, K.,** 2002. Maximization of Workspace Volume of 3-Dof Spatial Parallel Manipulators, *Journal of Mechanical Design*, **124**, 347-350.

[69] **Hay, A. M., and Snyman, J. A.**, 2004. Methodologies for the Optimal Design of Parallel Manipulators, *International Journal for Numerical Methods in Engineering*, **59**, 131-152.

[70] **Cortes, J., and Simeon, T.,** 2003. Probabilistic Motion Planning for Parallel Mechanisms, *Pr. of the 2003 IEEE International Conference on Robotics and Automation*, Taipei, September 14-19, 4354-4359.

[71] **Merlet, J. P.,** 1994. Trajectory Verification of Parallel Manipulators in the Workspace, *IEEE*, 2166-2171.

[72] **Dash, A. K., Chen, I. M., Yeo, S. H., and Yang, G.**, 2003. Singularity-Free Path Planning of Parallel Manipulators Using Clustering Algorithm and Line Geometry, *Proceedings of the 2003 IEEE International Conference on Robotics and Automation*, 761-766.

[73] **Toogood, R., Hao, H., and Wong, C.,** 1995. Robot Path Planning Using Genetic Algorithms, *IEEE*, 489-494.

[74] **Su, Y. X, Duan, B. Y., Peng, B, and Nan, R. D.,** 2003. Singularity Analysis of Fine-Tuning Stewart Platform for Large Radio Telescope Using Genetic Algorithm, *Mechatronics*, **13**, 413-425.

[75] **Su, Y. X, Duan, B. Y., and Zheng, C. H,** 2001. Genetic Design of Kinematically Optimal Fine Tuning Stewart Platform for Large Spherical Radio Telescope, *Mechatronics*, **11**, 821-835.

[76] **Su, Y. X, Zheng, C. H, and Duan, B. Y.,** 2002. Singularity Analysis of a 6-Dof Stewart Platform Using Genetic Algorithm, *IEEE*.

[77] **Yee, C., and Lim, K.,** 1997. Forward Kinematics Solution of Stewart Platform Using Neural Networks, *Neurocomputing*, **16**, 333-349.

[78] **Durali, M. and Shameli, E.**, 2004. Full Order Neural Velocity and Acceleration Observer for a General 6-6 Stewart Platform, *Proceedings of the 2004 IEEE International Conference on Networking, Sensing and Control*, 333-338.

[79] **Sreenivasan, S. V., Waldron, K. J., and Nanua, P.,** 1994. Closed-Form Direct Displacement Analysis of a 6-6 Stewart Platform, *Mechanism and Machine Theory*, **29**(6), 855-864.

[80] **Carretero, J. A, Podhorodeski, R. P., Nahon, M. A, and Gosselin, C. M,** 2000. Kinematic Analysis and Optimization of a New Three Degree-of-Freedom Spatial Parallel Manipulator, *Journal of Mechanical Design*, **122**, 17-24.

[81] **Anlı, E., Alp, H, Yurt, S. N, and Özkol, İ.,** 2004. The Stewart Platfrom Mechanism – A Review, *International Conference on Signal Processing*, İstanbul, December 17 – 19.

[82] **Yurt, S. N,** 2002. Kinematics, Dynamics and Control of the 6-3 Stewart Platform Mechanism, *PhD Thesis*, Istanbul Technical University, Institute of Science and Technology.

[83] **Blum, A**, 1992. Neural Networks in C++, John Wiley & Sons, Inc., NY.

[84] **Korn, G. A,** 1995. Neural Networks and Fuzzy Logic Control on Personal Computers and Workstations, The MIT Press, USA.

[85] **Leardi, R,** 2003. Nature-Inspired Methods in Chemometrics: Genetic Algorithms and Artificial Neural Networks, Elsevier B. V., Amsterdam.

[86] **Skapura, D. M,** 1996. Building Neural Networks, Addison-Wesley Publishing Company, ACM, USA.

[87] **Su, H. J., Dietmaier, P., and McCarthy, J. M.,** 2003. Trajectory Planning for Constrained Parallel Manipulators, *Journal of Mechanical Design*, **125**, 709-716.

[88] **Anlı, E., Alp, H., Yurt, S. N., and Özkol, İ.,** 2004. Uçuş Simülatörü Olarak Paralel Mekanizmalar, *Havacılıkta İleri Teknolojiler ve Uygulamaları Sempozyumu*, Hava Harp Okulu Komutanlığı, Yeşilyurt, İstanbul, December 9 – 10.

[89] **Alp, H., Anlı, E., Yurt, S. N., and Özkol, İ.,** 2004. Workspace Analysis of the 6 – 4 Stewart Platform Mechanism, *International Conference on Signal Processing*, İstanbul, December 17 – 19.

[90] **Anlı, E., Alp, H., Yurt, S. N., and Özkol, İ.,** 2005. Paralel Mekanizmaların Kinematiği, Dinamiği ve Çalışma Uzayı, *Hava Harp Okulu Havacılık ve Uzay Teknolojileri Dergisi*, **2**(1), 19-36.

[91] **Anlı, E., Yurt, S. N., and Özkol, İ.**, 2005. Forward Kinematics Analysis of a Purely Rotational Parallel Mechanism by Using Neural Networks, *Evolutionary and Deterministic Methods for Design, Optimization and Control with Application to Industrial and Societal Problems*, Munich, Germany, September 12 – 14.

Appendix A

Weights and Biases obtained after training with 17 123 purely rotational data patterns

First Hidden Layer Weight Matrix

```
 -2.2601 -2.6796  1.2012  1.3718   .3954   .8368

   .4936   .0433 -2.5339 -1.3392  2.3629  2.6085

  1.1678   .9856  1.2543  1.8623 -1.8344 -2.1278

   .5606   .4860   .5000   .3004   .4992   .5559

   .8936   .8206   .4596   .4332   .4688   .4868

  1.9139  -.6438  2.3143 -1.7695   .8796 -2.7131

  4.6864 -4.6418  1.6584 -2.2150  1.7732 -2.1134

  3.7118  3.8897 -1.3132 -2.7307 -2.2700  -.7648

  -.2114   .7603   .1142   .2762  -.3646  -.1463

  2.0407 -3.7149  2.1063 -3.3988  1.8193  -.9366
```

Second Hidden Layer Weight Matrix

```
  1.2869   .2265  -.3924   .4200   .1700  -.0163  -.0034  -.0627  -
.3690   .0385

  -.6598 -1.4112  1.0089   .2750  -.2900   .2692   .1981   .0019
.7457  -.2394

  1.1591   .3613   .2513  1.1244   .9264   .8807   .5297   .4152
.4443   .3858

  -.6316  1.3369 -1.1375   .3580   .2316  -.2280   .0231  -.3873  -
.1369  -.0493

   .1872  3.5628  -.6474  -.1235   .1289   .4286  -.1440   .5562
.1382   .4899

 -1.5951   .0468   .1453   .0340   .5062   .1477  -.0321  1.3066
.0471  -.0880

  -.9747  2.2572 -2.3317 -1.0659  -.2485   .0508   .0899   .3693  -
.0384  -.3205

  -.0415   .1258   .5636  1.3586  1.1643  1.0032  3.9041  -.2589
.7675  1.8867

  -.0598   .1414  -.1898  -.4140  -.3712   .7368  1.2600  -.1078
.0873  2.0678

   .1602  1.6700  -.5735  -.0178   .5677  -.0518   .6437 -5.4946
.2654  -.7616

  -.0295   .1580   .9175  -.0963  -.0244   .0665  5.9573  -.3722  -
.2584   .2432

  -.0276  -.0181   .0962 -1.5524 -1.0093  1.0607  1.2619  -.0539 -
1.4961  3.9706
```

Output Layer Weight Matrix

```
   .8068   .3415   .5492   .7903   .5998   .1023   .0065   .5008
.2111   .0145   .1592   .6246
```

```
   .2856   .4803   .9138   .7701   .3129   .2426   .6485   .2933
.1089   .7803   .9669   .5028

   .6908   .9694  1.2041  1.2899  1.3989   .3847  -.0862  1.0574
.4319  -.5029  -.4536  1.0357

  -.2026   .4452   .0757  -.5082 -2.7550  -.2233 -2.6828  -.1451  -
.0378   .0308  -.0072  -.0253

   .7822  -.1097  1.7729  -.0238  -.0009  -.9956  -.0500  -.0936  -
.0157  3.1602  -.0032   .0399

  -.0252   .1332  -.1915  -.0973   .0425   .0045   .0742  4.8171
.4054   .0046  3.8322  1.0728
```

First Hidden Layer Bias

```
  1.5009  -.1268   .2840   .4610   .1062  1.3501 -1.1199  2.0004
.4776  3.0972
```

Second Hidden Layer Bias

```
   .6419   .1559   .9217   .3009  -.1093  -.2338  -.4872   .9891
.0942   .1213   .5798  -.8002
```

Output Layer Bias

```
   .1516   .8808  1.5609   .6013  1.5676   .0447
```

Appendix B

Testing with one purely rotational data pattern

input leglengths
[0.9207 0.9280 0.7259 0.7394 0.9428 0.9619]

target values (x, y, z, γ, β, α)
[0.0000 0.0000 0.8000 -0.2443 -0.1048 -0.0174]

output values (x, y, z, γ, β, α) of the NN
[0.0000 0.0000 0.7999 -0.2481 -0.1052 -0.0169]
error: 0.004812 radians = 0.2758 degrees

output values (x, y, z, γ, β, α) of the NN after 1 loop
[0.0000 0.0000 0.7999 -0.2477 -0.1052 -0.0169]
error after 1 loop: 0.004294 radians = 0.2461 degrees

output values (x, y, z, γ, β, α) of the NN after 2 loops
[0.0000 0.0000 0.7999 -0.2473 -0.1052 -0.0170]
error after 2 loops: 0.003780 radians = 0.2166 degrees

output values (x, y, z, γ, β, α) of the NN after 3 loops
[0.0000 0.0000 0.7999 -0.2468 -0.1051 -0.0170]
error after 3 loops: 0.003269 radians = 0.1873 degrees

output values (x, y, z, γ, β, α) of the NN after 4 loops
[0.0000 0.0000 0.7999 -0.2464 -0.1051 -0.0171]
error after 4 loops: 0.002763 radians = 0.1583 degrees

output values (x, y, z, γ, β, α) of the NN after 5 loops
[0.0000 0.0000 0.7999 -0.2460 -0.1050 -0.0171]
error after 5 loops: 0.002261 radians = 0.1296 degrees

output values (x, y, z, γ, β, α) of the NN after 6 loops
[0.0000 0.0000 0.7999 -0.2456 -0.1050 -0.0172]
error after 6 loops: 0.001762 radians = 0.1010 degrees

output values (x, y, z, γ, β, α) of the NN after 7 loops
[0.0000 0.0000 0.7999 -0.2452 -0.1050 -0.0173]
error after 7 loops: 0.001267 radians = 0.0726 degrees

output values (x, y, z, γ, β, α) of the NN after 8 loops
[0.0000 0.0000 0.7999 -0.2448 -0.1049 -0.0173]
error after 8 loops: 0.0007765 radians = 0.0445 degrees

output values (x, y, z, γ, β, α) of the NN after 9 loops
[0.0000 0.0000 0.7999 -0.2444 -0.1049 -0.0174]
error after 9 loops: 0.0002894 radians = 0.0165 degrees

Appendix C

Weights and Biases obtained after training with 64 803 purely translational data patterns

First Hidden Layer Weight Matrix

```
  2.2848 -2.2796  -.8375  1.7390  -.5644   .6769

   .4456   .4715   .4507   .4717   .4871   .3789

  2.0082  -.4388  -.7273  1.0376   .8658  1.5098

   .8502   .6959   .7476   .4775   .8441   .8587

   .4006   .7820   .8796   .1108   .2252   .2692

   .1001   .9896   .8253  -.0897   .6118   .0568

  3.1664 -2.5892  -.8269  1.6114 -1.5374  1.3254

  1.0293   .5118 -3.3028  2.2435  2.0433 -2.6172

   .1440   .7010  2.1797 -1.9162 -1.9297  2.4958

  -.2139  2.1488   .1781  -.8787  2.2334  -.0424
```

Second Hidden Layer Weight Matrix

```
  -.5476  -.3794  -.1210  -.1072   .6540  -.3045 -1.2787   .0583  -
.1143   .6502

  1.0813  -.4159   .3590   .2429   .7741   .0280   .7956   .2588  -
.2577   .5825

   .6753   .0852   .0621   .7664   .5052   .7135  2.3278  -.0382
.0953  -.6843

   .3956 -1.2273   .7129  -.8073  -.4266 -1.1621  -.1275  4.8524 -
1.6471  2.2073

   .0246   .7349   .1665   .1639  -.1021   .8527  -.0887  1.8677 -
1.3170   .1419

  -.2615  -.3268   .6955  -.4418   .1281   .0562  -.1698   .6510 -
1.8465   .1161

 -1.1671  -.3268   .6815  -.5036  -.2420   .2481  1.4527   .0006  -
.2396 -1.6522

  -.2633  -.5721  1.5698  -.1647  -.5178  -.5975  7.4416   .2066  -
.7723 -1.1774

   .2414  -.1192  1.0083  -.1306   .1726   .6188  -.5127  -.8584
1.1888   .5202

  1.7307  -.7385  3.4646 -1.1143  -.5457 -1.0937 -2.7108 -1.6956  -
.5460  1.9499

  2.9540  1.3156   .7852   .9953   .9769   .9999  3.8631  -.1169  -
.1751 -3.1641

   .1706  -.4912  -.0751 -1.0136   .6292  -.2159  -.0301  1.1752 -
1.1895   .4281
```

Output Layer Weight Matrix

```
   .0404  -.0102   .0068  2.5269  2.6862   .2817  -.0190  -.0012  -
.1268  -.0333   .0141   .4882
```

```
  -.3875   .1322  1.1654   .0238  -.1196   .2144  1.3264  4.3098  -
.0145  -.0034  5.0249  -.1274

   .3811   .3759   .3764  -.1598   .7962   .4365  -.2005   .0122
.4560  3.0220   .0591  -.4793

   .7475   .2641   .3506   .0574   .3773   .7644   .5885   .4530
.5975   .5383   .7514   .6947

   .5485   .3436   .5498   .7373   .6702   .7592   .1598   .5715
.4503   .9349   .0803   .5534

   .5119   .2624   .2755   .1044   .8794   .7179   .2797   .1300
.5377   .9513   .8072   .2381
```

First Hidden Layer Bias

```
   .0028   .9139 -4.4732   .6085 -1.2634   .8437 -2.2270  -.7281 -
3.0973 -3.6551
```

Second Hidden Layer Bias

```
   .0088   .0506   .5482  -.8688   .2689  -.8267   .0902  -.7611
.2984  -.9804  1.2978  -.2265
```

Output Layer Bias

```
   .4814  -.2388  2.6414   .9656   .3335   .5908
```

Appendix D

Testing with one purely translational data pattern

input leglengths
[0.8683 0.8203 0.9629 0.7067 0.6774 0.9838]

target values (x, y, z, γ, β, α)
[-0.4000 0.0599 0.6600 0.0000 0.0000 0.0000]

output values (x, y, z, γ, β, α) of the NN
[-0.3973 0.0621 0.6680 0.0000 0.0000 0.0000]
error: 0.01285 meters

output values (x, y, z, γ, β, α) of the NN after 1 loop
[-0.3975 0.0619 0.6671 0.0000 0.0000 0.0000]
error after 1 loop: 0.01160 meters

output values (x, y, z, γ, β, α) of the NN after 2 loops
[-0.3977 0.0617 0.6663 0.0000 0.0000 0.0000]
error after 2 loops: 0.01035 meters

output values (x, y, z, γ, β, α) of the NN after 3 loops
[-0.3979 0.0615 0.6655 0.0000 0.0000 0.0000]
error after 3 loops: 0.009098 meters

output values (x, y, z, γ, β, α) of the NN after 4 loops
[-0.3982 0.0613 0.6646 0.0000 0.0000 0.0000]
error after 4 loops: 0.007841 meters

output values (x, y, z, γ, β, α) of the NN after 5 loops
[-0.3984 0.0611 0.6638 0.0000 0.0000 0.0000]
error after 5 loops: 0.006582 meters

output values (x, y, z, γ, β, α) of the NN after 6 loops
[-0.3986 0.0609 0.6630 0.0000 0.0000 0.0000]
error after 6 loops: 0.005321 meters

output values (x, y, z, γ, β, α) of the NN after 7 loops
[-0.3989 0.0607 0.6621 0.0000 0.0000 0.0000]
error after 7 loops: 0.004057 meters

output values (x, y, z, γ, β, α) of the NN after 8 loops
[-0.3991 0.0605 0.6613 0.0000 0.0000 0.0000]
error after 8 loops: 0.002790 meters

output values (x, y, z, γ, β, α) of the NN after 9 loops
[-0.3994 0.0603 0.6605 0.0000 0.0000 0.0000]
error after 9 loops: 0.001520 meters

Appendix E

Weights and Biases obtained after training with 53 415 general data patterns

First Hidden Layer Weight Matrix

```
  3.6259 -2.1302   .7559   .0120  6.2939 -4.3281

 -3.8651  2.3554 -6.4368  5.9690 -1.4524  -.7063

   .3167  -.9281  1.6598  1.6274 -2.3615 -1.3223

   .5208  -.9000 -1.2047 -1.3411   .9057  2.2035

  2.5195  1.6570  -.0725 -1.8953  -.3787 -1.7292

 -1.0011  7.0310 -1.0016 -1.1510 -1.1023 -1.3894

 -2.8163  4.0935 -5.7037  6.4583   .5752  1.3168

 -2.1831  3.5493   .2304   .8064 -4.0034  6.0951

  1.8104  -.8490 -3.1767  2.7304 -2.9667  2.6783

  4.7946 -2.9631 -1.3254  1.1716 -1.4980   .5095
```

Second Hidden Layer Weight Matrix

```
   .1208  -.0783  -.2894  -.6588   .4901 -4.4586  -.1007  -.1119
.0019 -1.7425

 .8564 -1.4678  -.0422  -.1893  -.0567  1.5183  -.7128   .7971
.1228  -.0421

  1.0865  -.7072  -.2395  -.3571  -.0146  -.3612   .6614  -.8477
.1104   .0528

  -.0302  -.0087   .2959   .4673 -3.0004 -2.8223  -.0052   .1058 -
1.3704   .7543

   .1697  -.1517  1.0428  -.6921  -.0121   .2068   .1601   .1639  -
.0068  -.0047

   .7701   .6652  -.3994  -.4199  -.1294  -.5530  -.7034 -1.0847
.0196   .1070

   .0187  -.0224  -.6177 -1.1816   .3060  -.0356  -.0461   .1381  -
.4141 -2.8201

   .8345   .7197   .0445  -.0881  -.0615  -.7246  1.2633   .8741  -
.0342  -.0263

   .1538   .1697   .0277 -1.3987 -1.2066   .8088  -.0094   .6803  -
.6613  -.6013

  -.0078  -.0422  1.3632  2.6647  -.6335 -7.3095   .1143  -.4278
3.5496 11.4514

  -.2451   .1797 -2.7169  2.3304   .0626 -3.5800  -.1538  -.2508  -
.0283  -.0889

  -.0277  -.0083  -.0610  -.1664  1.6016  -.2513   .1286  -.0702
.9634  -.7256
```

Output Layer Weight Matrix

```
   .2839 -2.6880  2.6152   .6194  -.3238  2.5236  -.0039  2.5231  -
.0969   .1560  -.4705  -.0430
```

-18.1077 -.2040 -.1377 1.0088   .7433   .1862 -.8401 -.1566
.0525 10.6573   .2182 1.0814

  2.5153 4.2258 3.8503   .3312 1.1662 -4.0188 -.1118 3.6397
.0232 1.7731 -1.8284 -.0530

  -.0558 -.0577 -.1016 -.0766 22.8648   .0992   .0466 -.0166 -
.0472 -.2239-21.8040 -.2029

   .6281   .0970   .1288 18.6834   .0569   .0532   .0046 -.0542
.1849 -.0667   .2898-24.1022

   .1745 3.1218 2.5353   .1820 -.5830 2.7758 -.1117 -2.8697
.6416   .1619   .8563 2.9049

First Hidden Layer Bias
-4.6781 2.5383 1.3987 -.7582   .1198 -9.1029 -2.3290 -4.8583 -
1.0250 -1.4879

Second Hidden Layer Bias
-3.3062   .4886   .3309   .5101 1.6366 -1.1818 -3.3036 -1.5270 -
1.1629 8.9167 1.8295 1.8470

Output Layer Bias
  -.5678 3.7603 -2.0908 -.0444 4.4255 -2.6517

Appendix F

Testing with one general data pattern

input leglengths
[0.7134 0.9120 0.8836 0.8669 0.8101 0.8667]

target values (x, y, z, γ, β, α)
[-0.0535 -0.1442 0.7515 0.0491 0.0830 -0.1652]

output values (x, y, z, γ, β, α) of the NN
[-0.0490 -0.1458 0.7545 0.0496 0.0770 -0.1598]
error: 0.02097

output values (x, y, z, γ, β, α) of the NN after 1 loop
[-0.0494 -0.1456 0.7541 0.0495 0.0776 -0.1601]
error after 1 loop: 0.01900

output values (x, y, z, γ, β, α) of the NN after 2 loops
[-0.0498 -0.1454 0.7537 0.0495 0.0782 -0.1604]
error after 2 loops: 0.01704

output values (x, y, z, γ, β, α) of the NN after 3 loops
[-0.0503 -0.1453 0.7534 0.0494 0.0789 -0.1607]
error after 3 loops: 0.01509

output values (x, y, z, γ, β, α) of the NN after 4 loops
[-0.0507 -0.1451 0.7530 0.0494 0.0795 -0.1610]
error after 4 loops: 0.01315

output values (x, y, z, γ, β, α) of the NN after 5 loops
[-0.0511 -0.1449 0.7527 0.0493 0.0801 -0.1614]
error after 5 loops: 0.01121

output values (x, y, z, γ, β, α) of the NN after 6 loops
[-0.0515 -0.1447 0.7524 0.0493 0.0807 -0.1617]
error after 6 loops: 0.009289

output values (x, y, z, γ, β, α) of the NN after 7 loops
[-0.0502 -0.1446 0.7521 0.0492 0.0814 -0.1620]
error after 7 loops: 0.007368

output values (x, y, z, γ, β, α) of the NN after 8 loops
[-0.0524 -0.1444 0.7518 0.0492 0.082 -0.1624]
error after 8 loops: 0.005455

output values (x, y, z, γ, β, α) of the NN after 9 loops
[-0.0528 -0.1442 0.7514 0.0491 0.0826 -0.1628]
error after 9 loops: 0.003551

BIOGRAPHY

Elmas Anlı was born in İstanbul in 1981. She graduated from Uskudar American Academy in 1999. She received her BSc degree in 2003, from Istanbul Technical University, Faculty of Aeronautical and Astronautical Engineering, Department of Astronautical Engineering and ranked 1st in her class during graduation. She started her graduate education at Istanbul Technical University, Insitute of Science and Technology, Department of Aeronautical and Astronautical Engineering in 2003. She has been working as a Research Assistant in the Department of Aeronautical Engineering since 2004.